RADC-TR-90-52
Final Technical Report
May 1990

DISTRIBUTED OPERATING SYSTEMS INTEROPERABILITY RESEARCH PROJECT

Dynamics Research Corporation

Sponsored by
Strategic Defense Initiative Office

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Strategic Defense Initiative Office or the U.S. Government.

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

DISTRIBUTED OPERATING SYSTEMS INTEROPERABILITY RESEARCH PROJECT

Prof John A Stankovic
Prof David R Cheriton
Prof Partha Dasgupta
Mr Mike Dean
Prof E Douglas Jensen
Dr Thomas Raeuchle
Dr Richard C Schantz

Contractor: Dynamics Research Corporation
Contract Number: F19628-86-D-0006
Effective Date of Contract: 15 May 1987
Contract Expiration Date: 30 September 1987
Short Title of Work: Distributed Operating Systems Interoperability
Period of Work Covered: Jun 87 - Sep 87

Principal Investigator: Prof John A Stankovic

RADC Project Engineer: Thomas F Lawrence
Phone: (315) 330-2158

This research was supported by the Strategic Defense Initiative Office of the Department of Defense and was monitored by Thomas F Lawrence, RADC (COTD), Griffiss AFB NY 13441-5700 under Contract F19628-86-D-0006.

REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1990	Final Jun 87 to Sep 87

4. TITLE AND SUBTITLE

DISTRIBUTED OPERATING SYSTEMS INTEROPERABILITY RESEARCH PROJECT

5. FUNDING NUMBERS

C - F19628-86-D-0006
PE - 63223C
PR - 2300
TA - 02
WU - 31

6. AUTHOR(S)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Dynamics Research Corporation
60 Frontage Road
Andover MA 01810

8. PERFORMING ORGANIZATION REPORT NUMBER

E-13638U

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Strategic Defense Initiative
Office, Office of the Secretary of Defense
Wash DC 20301-7100

Rome Air Development Center (COTD)
Griffiss AFB NY 13441-5700

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RADC-TR-90-52

11. SUPPLEMENTARY NOTES

RADC Project Engineer: THomas F. Lawrence/COTD/(315) 330-2158

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report presents the results of the Distributed Operating System Interoperability Research project sponsored by the Air Force Rome Air Development Center (RADC) from June through September of 1987. The goal was to identify guidelines for developing a future integrated Distributed Operating System (DOS) to support a large Battle Management/Command, Control and Communication (BM/C3) system. The project participants drew upon their experience with existing DOSs to determine the key issues involved in designing a future DOS.

After comparing BM/C3 requirements and the status of current DOS research, we concluded that no current DOS designs adequately address BM/C3 requirements and additional research should be focused on meeting BM/C3 needs. In particular, significant research should be directed toward evolving a DOS design that can simultaneously provide: Evolvability, Fault Tolerance, Multi-Level Security (Continued on reverse)

14. SUBJECT TERMS	15. NUMBER OF PAGES
Distributed operating systems, distributed systems, system architecture, interoperability	136
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Item 13 Continued:

and Real-Time Processing Support.

Properly supporting BM/C3 applications requires new methods of system-wide resource management. Generally, existing DOS designs have not significantly evolved beyond their time-sharing network origins. As a result, they still manage system resources to achieve maximum system throughput, with some degree of fairness so that no task has to wait forever. Furthermore, each system element is managed relatively independently. This is not adequate for BM/C3 applications. A BM/C3 DOS must coordinate the use of all system resources to meet real-time response, fault tolerance and multi-level security objectives. This requires new strategies for managing each of the system components.

FOREWORD

This report presents the results of the Distributed Operating System Interoperability Research project sponsored by the Air Force Rome Air Development Center (RADC) from June through September of 1987. The goal was to identify guidelines for developing a future integrated Distributed Operating System (DOS) to support a large Battle Management/Command, Control and Communication (BM/C3) system. The project participants drew upon their experience with existing DOSs to determine the key issues involved in designing a future DOS.

This document is a final technical report fulfilling CDRL Items 111 and 112 of the task order for Technical and Engineering Management Support (TEMS) Task 0441, the Distributed Operating Systems Interoperability Research Project under contract number F19628-86-D-0006.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

SECTION 1. - INTRODUCTION

1.1 BACKGROUND

One long term objective of Rome Air Development Center (RADC) is to design a highly integrated Distributed Operating System (DOS) to support a large Battle Management/ Command, Control and Communications (BM/C3) system. This integrated DOS will manage a complex computing base made up of clusters of heterogeneous computers. As the system evolves to meet new threats, it will contain a mixture of new advanced components and older, more established technology. To achieve high-level system objectives such as graceful degradation in the face of failures, or security against data and software tampering, all parts of the system must work together harmoniously at all times. Therefore, RADC is particularly interested in distributed operating system design approaches that will enable building a large, evolvable, fault tolerant, secure, and highly integrated, real-time BM/C3 system.

Currently, RADC is funding several research projects which are actively investigating different aspects of a DOS. Each of these research groups is building a DOS whose design and implementation is geared to supporting a particular BM/C3 requirement, such as real-time responsiveness, fault tolerance or security. Research in these projects has reached a point where the principal investigators can provide RADC with recommendations for promising research based upon experience gained from experimental implementations.

To draw upon this experience, RADC established the DOS Interoperability Research Project (DOSIRP), described in this report. As participants in this project, we were asked to focus on methods for providing interoperability (Refer to Section 1.2) between clusters of heterogeneous computers in a BM/C3 environment. Our objective was to provide DOS researchers and RADC research managers with an understanding of what the "BM/C3 DOS of the future" might look like, based on current experience.

RADC is also establishing the Distributed System Evaluation Environment (DISE) to demonstrate and integrate various distributed system research projects. Although DOSIRP indirectly supports this effort, it was not our intent to

determine what it would take to tie existing systems together. Other efforts have been made in that regard. Rather, our objective was to determine the essential characteristics of a future DOS that would efficiently support interoperability within a BM/C3 environment.

We attempted to establish a set of guidelines for structuring future systems based on experience gained from existing systems. Since distributed operating systems research for BM/C3 applications is still in its infancy, in many areas only tentative recommendations for future research directions and approaches were possible. However, there were several areas where we felt we had enough experience to establish preliminary guidelines. We attempted to identify all of these areas and focus on the most promising and critical ones. What emerged from this effort is a basic system architecture, and a set of design guidelines and recommended research approaches to address the issues regarding access control, addressing, authentication, communications, naming, storage, synchronization, system-wide resource management and timing. These results provide a basis for evaluating current technology and for focusing future research efforts.

1.2 INTEROPERABILITY

Our charter was to investigate DOS design issues that relate to providing interoperability between many heterogeneous computer resources in a BM/C3 environment. Our first task was to define what we mean by interoperability.

We consider interoperability to mean the efficient interaction between clusters of computers with very little translation overhead. Interoperability also implies the ability to do system-wide resource management and to provide flexible, but controlled access to any part of the system. The degree of interoperability of a system is not quantifiable, but is dependent on two factors:

1) the ease of integration of new components, and
2) the efficiency of the resulting distributed system.

It is useful to think of interoperability as a measure of both the efficiency of interaction and the ease of modifying the system to allow new interactions. As an analogy, consider the idea of portability. One could define a software module as portable on the basis that it can be made to run in a new machine environment in

"a short time". But how short a time? Two seconds is clearly acceptable, two weeks may be reasonable, but two years is surely unacceptable. Similarly, the degree of interoperability should be gauged by the effort it takes to integrate a new component into the system and, once it is in the system, by the ratio of productive work to total activity.

Many existing systems claim to be interoperable with other systems, but current techniques rarely achieve the degree of interoperability needed for BM/C3 applications. It is currently a very difficult and time consuming process to add system components, and the "glue" software needed to make the pieces "interoperable" is costly in terms of execution time. DOS researchers need to develop more sophisticated techniques to achieve a high degree of interoperability. It is also important to minimize arbitrary differences in expressing the same concept, because these cloud the real issues and unnecessarily inhibit interoperability. However, in a field as immature as DOSs, it is not at all clear which differences are arbitrary.

In most areas of DOS research, especially for BM/C3 applications, it is too early to establish rigid interface standards. It is important to maintain the diversity necessary to explore new concepts. However, it is not too early to identify areas where interim approaches can be developed, and to define research needed to explore potential standards. To this end, we have proposed a system architecture and have identified nine essential DOS components to facilitate interoperability. The main text of this report explains these nine areas, discussing known results, opinions, open questions and research recommendations.

1.3 OVERVIEW OF THIS REPORT

Section 2 discusses BM/C3 system requirements that impact the DOS design. These requirements impact DOS design so significantly that much of the current DOS research outside the BM/C3 community is not applicable. We attempt to address this issue throughout the report. Section 2.3 ties together three areas of research: DOSs, the object-oriented paradigm and BM/C3 systems, and describes one way to view a BM/C3 system from an object-oriented standpoint. In Section 3 we recommend a high-level system architecture such as the one described. This architecture includes a description of the system model (Section 3.1), the basic

system structure (Section 3.2), the attributes of the basic components (Sections 3.3) and the semantics of objects (Section 3.4). Section 4 identifies and describes nine components of a DOS we deem critical to achieving interoperability. These components provide the core functionality needed in BM/C3 DOSs. Section 5 summarizes the conclusions and recommendations that emerged from the project and outlines a research plan.

SECTION 2. - REQUIREMENTS FOR A BM/C3 DOS

2.1 OVERVIEW

A BM/C3 system must support many applications, ranging from real-time control to data archiving and administrative functions. To operate effectively, the BM/C3 system must contain many clusters of computers functioning in concert. These clusters will have radically different objectives and consequently will have distinct operating systems tailored to maximize effectiveness in different areas. For example, real-time control may be forced to sacrifice security to achieve speed, while system-status monitoring may have to trade speed for a high level of security. In spite of these conflicting design objectives, the various modules of the BM/C3 system must interact effectively to achieve overall system goals.

Most DOS research does not explicitly deal with any of these requirements, because attention is focused on the largest and most popular commercial applications. Some researchers have studied some of these requirements, but no one has dealt with all of them. This section discusses requirements that are unique to the BM/C3 environment. Appendix B goes into more detail about some BM/C3 characteristics and the implications for DOS design.

2.2 KEY ATTRIBUTES OF A BM/C3 DOS

We specifically focused on the attributes a DOS should have to support in a BM/C3 system. The major factors considered included:

- Evolvability
- Fault tolerance
- Multi-level security
- Real-time processing support

These key issues are discussed below.

2.2.1 Evolvability

Once a BM/C3 system is deployed, it may remain in operation twenty years or more, with new hardware and software being added, modified or deleted to extend it's lifetime as new environments and threats develop. It is essential that a large BM/C3 system be able to evolve and change incrementally since it would be infeasible to replace an out-of-date system by a completely new state-of-the-art system.

There are many reasons why the ability to evolve gracefully is important to a large BM/C3 system. When such a system is deployed, there is a very slim chance that it will work perfectly on the first try. Design flaws will need to be corrected without having to redesign large portions of the system. Furthermore, it is inconceivable that designers will be able to anticipate every possible future threat. As new threats are encountered, the system must be able to change accordingly, to defend against these new threats. This might involve a change in mission requirements and system policy which could trigger a re-evaluation and modification of the protocols and mechanisms currently being implemented. Finally, the system should be able to take advantage of newer technology.

The necessary design evolution can be accomplished much more smoothly if, early in the development process, the system is decomposed into more manageable pieces (i.e., into modules or subsystems). Once decomposed, the subsystems should be structured such that changes reflecting changes in the system policy or technology upgrades can be made with minimal effort. There are two approaches to structuring these modules which seem worthwhile: information hiding and policy/mechanism separation.

In the information hiding approach, the module interfaces are clearly defined and strictly enforced so modules can be added, or the internals of a module modified, as long as the correct interface remains (See Figure 2-1). In this way, the impact of design changes can be confined to a single subsystem, and design evolution is made much simpler. On the other hand, too many rigid interface constraints or a poor partitioning into subsystems can severely inhibit design evolution. Therefore, it is extremely important to define interfaces that will not

unnecessarily inhibit evolution. One well-known approach used to achieve information hiding is the data abstraction technique.

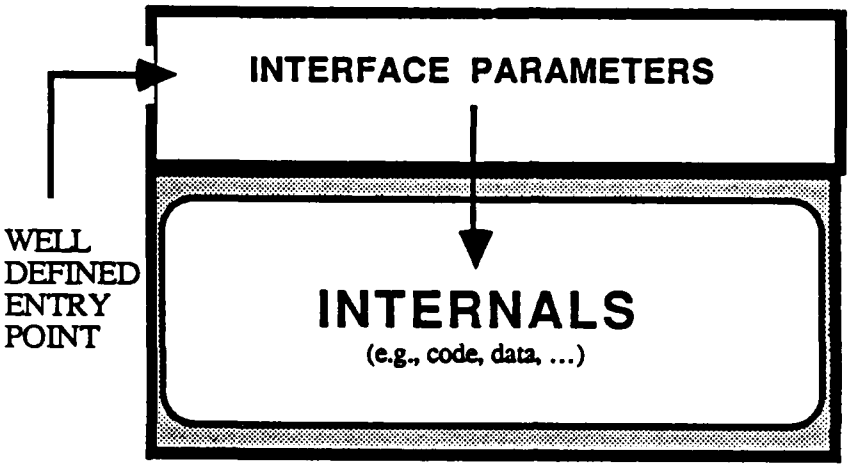


Figure 2-1 Information Hiding Approach -
By adhering to the interface, the internals can take on any structure and still be interoperable with other modules.

Data abstraction focuses on encapsulating data structures and the procedures that operate on them so that implementation details are compartmentalized. This limits the impact of data representation changes. Each major structure and its associated operations is viewed from the outside as an abstract data type with specified behavior. The internal data representation is visible only to the procedures that directly modify the data. This is analogous to the way most high level languages treat their built in data types. For example, in ADA a programmer does not know the storage format of integers; he only knows their behavior when he does things like add and subtract them.

Another approach is to attempt to separate the policy from the mechanisms which support the policy (See Figure 2-2). Ultimately, a set of universal mechanisms is developed from which any policy can be supported. However, the more realistic approach would be to find a set (or sets) of mechanisms which support many (vs. all) policies. The advantage of this approach is that any changes in the existing policy merely requires the current set of mechanisms to be replaced by a new or modified set of mechanisms which work together to meet the new

policy requirements. The difficulty arises in trying to determine what the appropriate set of mechanisms is, such that any policy can be supported.

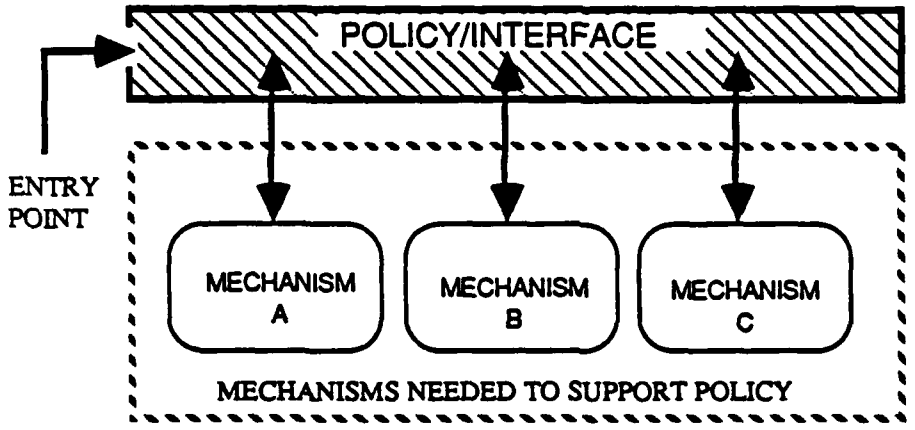


Figure 2-2 Policy/Mechanism Separation Approach -

If the policy changes, the set of supporting mechanisms may also change, but the mechanisms themselves do not.

It is likely that future BM/C3 systems will utilize object-oriented design because it facilitates design evolution and it readily supports information hiding, data abstraction, and policy/mechanism separation Furthermore, object- oriented design closely matches a system architect's view of the system and is easily adaptable to distributed processing. For this reason, we were explicitly asked to consider the object model. Section 3. discusses this model in more detail.

In a BM/C3 system, designing for other desired attributes often simplifies system evolution. For instance, to achieve a certain degree of resiliency, the network may be designed to reconfigure itself around failures. By possessing the ability to reconfigure, the network is in fact deleting old nodes and adding new nodes, which also solves some of the problems of evolvability.

However, there are some attributes which further complicate the evolvability process. For example, when a module is added, deleted, or modified, the security of the system must be verified for every possible reconfiguration. In the case of a large BM/C3 system, this could be a major undertaking. Another problem occurs

if a change is made to a component where the security mechanisms usually reside. In this case it is critical that the change only be made by a trusted agent. Furthermore, in a BM/C3 system these changes must be made without halting system operation (non-stop or continuous operation).

2.2.2 Fault Tolerance

There are many factors which affect the choice of a fault tolerance scheme for a BM/C3 system. The four major factors include: the size of the engineering problem, the several modes of operation (peacetime, test, alert and battle mode), space and power limitations, and the need for the system to be evolvable. A fault tolerance scheme that addresses all of these problems must be found without sacrificing the needed real-time performance and security. Fault tolerance is already a difficult problem in existing large-scale systems, but the complexity of the problem increases dramatically when coupled with the other BM/C3 requirements. In this section we describe the four major factors affecting the design of a fault tolerant BM/C3 system. The four factors will be discussed separately, but keep in mind that each of these factors has an impact on every other factor.

Fault tolerance has been a topic of research for over two decades now, however, other BM/C3 requirements eliminate the possibility of using or extending many existing techniques. Existing fault tolerance solutions cannot simply be scaled up to suit this new problem of designing a fault tolerant BM/C3 system. First, power and space limitations on space-based platforms eliminate the use of many existing redundancy and high overhead techniques. Second, there are different modes of operation, e.g., peacetime and battle mode. A scheme which accounts for entirely different set of needs/requirements (which will be discussed in more detail below) must be developed, while providing continuous and correct behavior. Third, the scheme which is finally adopted must be readily changeable to adapt to new environments. Below, we discuss the issues which make the existing techniques inapplicable in a BM/C3 environment.

The sheer magnitude of proposed BM/C3 systems pose new problems for hardware and software engineers. The number of nodes and the amount of software involved requires a decomposition of the problem into smaller, more manageable pieces which can later be combined to provide the total functionality

strived for. What is needed is a system design that is easy to analyze and test so that there is more confidence/assurance in the behavior of the system when it is deployed. In addition, software fault tolerance schemes should be pursued rather than working towards the development of error-free code, since the likelihood of writing over ten-million lines of perfect code is very small.

There are several possible modes that the system could be operating in and each of these calls for a very different fault tolerance strategy. For example, in the peacetime mode, the main function of the system is surveillance. The types of faults that are likely to occur during this phase are faults due to natural causes, such as mechanical wearout or radiation effects. The main objective is to ensure that the system remains functional for its intended lifetime (of twenty years or more) with as little need for human intervention as possible (since maintenance and repair periods are the most vulnerable to breaches in security). What seems promising are automated maintenance and self-repair strategies.

During the battle mode, these high bandwidth schemes become useless since the communications bandwidth will likely become degraded due to jamming and nuclear effects. Also, hardware failures caused by battle damage are likely to be clustered in time and space instead of being randomly distributed. What is needed is a scheme that will tolerate multiple, simultaneous, correlated faults within stringent timing constraints. In both modes of operation, there is the added problem of limited access to faulty components that need repair (i.e., space-based components). Above all, the fault tolerance scheme must not interfere with the execution of critical functions.

To devise a scheme which ensures a meaningful response or behavior to these faults, the system architect must analyze all conceivable threats to the system and then identify the possible fault groups. Depending upon the fault tolerance needs of the application, the situation, and the nature of the fault, a fault tolerance strategy is devised for each function in the system and each fault group. The development of an adequate fault tolerance model is very difficult since it is virtually impossible to anticipate what the future threats may be, and in turn, what kinds of faults these threats may induce. Current models assume fail-stop behavior, and it must be determined whether this is realistic in a BM/C3 environment. Additionally, technology is advancing fairly rapidly in the computer world, and it is highly unrealistic to expect a total redesign and re-implementation of a BM/C3 system

when newer, improved technology becomes available. This necessitates a temporary solution to the fault tolerance problem which has the capability of being extended and modified as the system evolves in response to changes in the mission requirements, technology upgrades and new threats.

The redundancy found in distributed BM/C3 systems provides a good basis for building fault tolerant systems. Many DOS algorithms are themselves replicated (e.g., see Section 4.1 on system-wide resource management) and are inherently decentralized, thus providing a degree of fault tolerance.

To allow the applications designer to tailor the failure behavior of an object to accommodate different fault tolerance requirements the DOS should support a variety of fault tolerance mechanisms. As a minimum, the DOS should be able to detect certain types of faults and notify the application. In addition, some of the fault tolerance strategy should be implemented at the application layer through fault recovery code. In other situations, there should be a mechanism for communicating fault tolerance requirements from the application layer to the DOS. This information could take the form of parameters that the application specifies to the DOS to notify it of how many replicas of a particular object to maintain, and how to distribute them through the system. Just how one implements the above strategies must be investigated.

2.2.3 Security

It is essential to ensure the integrity of any system. This is especially true of a BM/C3 system in which compromise of the system has serious consequences. For this reason, it is critical that data, hardware and software modules be protected from inadvertent or malicious misuse by users. Users can refer to people or other computing systems external to the system, or it can refer to objects internal to the system (i.e., protecting objects from each other). Security involves the authentication of users requesting information or services and a check of the requestors access rights or privileges to this information or service. It also involves the flow of information (mandatory security). We are more interested in the former case although both mandatory and discretionary policies are needed in a BM/C3 system.

Some parts of a BM/C3 system must deal with information that is highly classified, such as intelligence information. Other parts of the system must deal with information that is not so sensitive, but is still classified. Large portions of the system will not need to deal with classified information at all.

The DOS must provide mechanisms to safeguard information, but still permit interaction between all parts of the system. If all of the system were cleared for access to all information (system-high), then physical security would be enough to protect the information. Or if a portion of the system could be isolated then it would be easy to contain information within it. The goal is to provide adequate interoperability while maintaining multiple levels of security in different parts of the system. Some sources and causes of information leakage or tampering can be worms, viruses, time bombs, Trojan horses, and ferreting activity.

The tradeoffs between real-time performance, fault tolerance and security are not well understood but need to be determined so that security considerations can be addressed during the design phase of the system rather than as an applique. In some ways it would be advantageous to place the security mechanisms at a low-level. This would increase the system efficiency for those needing the security, but may not be acceptable for real-time applications where any additional overhead is unacceptable. Another problem occurs during maintenance and update. For a system that must operate for twenty years or more, maintenance and updates are inevitable. If security mechanisms only work at the DOS level and above, then during maintenance and updates the system will be vulnerable to security threats. This is because maintenance and updates are usually done below the DOS level where there are no security mechanisms. Thus, it is essential that highly cleared personnel and a trusted environment is established.

The BM/C3 systems, as mentioned above, usually have several modes of operation. Each mode may have very different security policies. For example, during the peacetime mode, security of the system is of utmost importance since data tampering, compromised software, information leakage and any other breach in security could cause catastrophic consequences. During the battle mode, some of the security may have to be traded off for performance and/or fault tolerance. Again, these tradeoffs are not well understood and must be studied so that security considerations can be addressed.

2.2.4 Real-Time Processing Support

In a BM/C3 system, there are many time-critical tasks, both periodic and aperiodic. By real-time processing, we mean the ability to complete a task by some time T, where T may be the time at which a catastrophic event may occur, or a time at which the task results would be useless. Examples of when completion of a task before it's deadline T is important include (1) capturing data before sensor buffers overflow and (2) sending command signals in time to be effective.

The failure to meet real-time deadlines impacts the correctness of the system's behavior and may have catastrophic consequences. Special mechanisms must be included in the DOS to provide real-time processing support. However, systems with hard real-time constraints add significant complexity over conventional DOSs. Conventional DOSs usually use priority scheduling/schemes and do not ensure that deadlines are met. Standard priority scheduling methods are not enough for BM/C3. Scheduling techniques must be able to integrate the timing constraints of multiple parts of an application, of the DOS and of the communications subsystem. In a large, geographically distributed system where propagation delays are non-negligible, the system must have the ability to generate some total ordering for scheduling based upon incomplete global information. Also, application programs need to have mechanisms for communicating deadlines to the DOS, and the DOS needs to have mechanisms to guarantee meeting deadlines.

2.3 AN OBJECT ORIENTED VIEW OF A BM/C3 SYSTEM

This section ties three areas of research together which, jointly, will form the focus of the remainder of the report. The three areas are distributed systems, the object-oriented paradigm and BM/C3 systems. What is presented here is one example of how a BM/C3 subsystem might be viewed from an object-oriented standpoint. Preceding this example is a brief overview of the objectives of the Strategic Defense System (SDS) and it's BM/C3 subsystem.

2.3.1 Objectives of the SDS and it's BM/C3 Subsystem

The main goal of the SDS is to be able to react to, and eliminate, any missile threat within strict time windows. The SDS will consist of global battle managers, regional/area battle managers, space-based weapons platforms, space-based sensors, airborne sensors, ground sites and BM/C3 platforms. The estimated constellation size, which is strongly dependent on the weapons and threat characteristics, has varied considerably over the past and has numbered from as few as 50 weapons satellites, to as many as 2000 sites. Regardless of the constellation size, the components will be geographically distributed, and will probably contain systems implemented with older technology, as well as state-of-the art technology. These distributed components will be organized into a hierarchy of control and need to cooperate and act in synchrony to protect against the threats through all phases of the (boost, post-boost, midcourse and terminal) defense. [DRC86].

The backbone of the SDS is the BM/C3 subsystem. It's main objective is to provide an effective defense system under all conflict conditions. In order to achieve this goal the BM/C3 system must support the following key functions: (1) ensure the availability of the SDS elements in support of positive and distributed control (i.e., ensuring that there are no missing links in the chain of command at any moment), (2) provide for secure processing and transfer of information to support informed decision making, (3) provide enhanced survivability through system evolution and robust communications and (4) provide trusted software that supports the most efficient and effective employment of the SDS.

From a system designers point-of-view, the above functions could be decomposed and viewed as the algorithms associated with information acquisition and control, and the implementation technologies needed to support them. The algorithms for information acquisition and control can be decomposed and grouped into three main classes: data processing algorithms, decision algorithms, and control algorithms. The data processing algorithms collect and process sensor information. The decision algorithms assess the current situation, based on the threat data, and also assigns the weapons and sensors to oncoming targets. The control algorithms implement the policies set by the decision algorithms.

The three main implementation technologies needed to support the above algorithms include: communications, data management and computation. Communications technology allows the transfer of information needed by the BM node, and is also a necessary part of network management (since real-time reconfiguration is mainly a reassignment of communications paths between nodes). Data management technology supports the transfer of data to the appropriate nodes and the maintenance of the (defense and threat) databases. Computation technology will ensure the availability of adequate processing power to achieve the required overall system response.

It is essential to choose a foundation that will accommodate any future extensions, modifications and additions to the implemented system. This is especially true of the BM/C3 subsystem (and the SDS as a whole) since researchers and system designers are basing their decisions on the results and experience gained in new and quickly-changing fields. The foundation chosen for our study is a distributed, object-oriented system. In particular, the distributed operating system support necessary to achieve interoperability within a large-scale, distributed, object-oriented system is being investigated.

2.3.2 Why a Heterogeneous, Distributed, Object-oriented System?

The BM/C3 system is envisioned as a large, geographically distributed system consisting of thousands of nodes. These nodes are likely to contain different machine types and software components for their specialized functionality or speed (or simply because they exist and are known to work). The physical distribution of the system and the need for graceful degradation during failures naturally leads to the distributed system approach. These distributed computers must interact in a cooperative manner to implement a significant number of decentralized control algorithms. The complexity, large-scale nature, heterogeneity and anticipated evolution of the system favors an object-oriented paradigm. Hence, we have chosen to focus our study on the distributed operating system support necessary to achieve interoperability between heterogeneous, distributed, object-oriented, computing systems.

It must not be forgotten that object-oriented systems can have significant overhead associated with message passing implementations. For our study, the advantages of an object-oriented system are presumed to outweigh the disadvantages.

2.3.3 How One Can View Such a System

The object-oriented paradigm allows the system developer to view the system as it would appear in reality - as a group of objects which interact with each other and have individual characteristics and behaviors. It allows a large, complex object to be modularized into smaller, more manageable pieces which are easier to work with. Once modularized, and assuming the modules are chosen properly, there seems to be one of two main routes one can follow. The first is specifying and adhering to interfaces and protocols (as in the information hiding and data abstraction approaches mentioned in Section 2.2.1). If the interfaces are well-defined and adhered to, the individual modules can be easily joined after development and testing to construct the original complex object. The second route is to separate the policy from the mechanisms and standardize on the mechanisms (see Section 2.2.1). This allows the policy to change without having to change the mechanisms.

The major advantage of the first route is that custom software and hardware can be developed without worrying about the different input and output formats of other modules since the format will now be the same. The disadvantage of this approach is that, initially, the interface may be too general (in an attempt to accommodate evolution) and have a large overhead associated with it. Furthermore, this very interface may become overly restrictive if new technology and new solutions arise which were totally unforeseen (which is usually what happens). The major advantage of the second approach is that once you have the appropriate mechanisms in place, the policy can be changed and this change in policy simply reflects a regrouping of the mechanisms such that they cooperate to support the new policy.

In a BM/C3 system, at a high level of abstraction, the objects may include one or more of the following subobjects: a status monitor, a data correlator and

discriminator, a situation assessor, and/or a planner and controller. Note that a status monitor for a sensor will not have the same characteristics and responsibilities of a status monitor for a battle manager. Furthermore, some subobjects may not even be found on other objects (see Figure 2-3). Each of the subobjects will have an associated database (see Figure 2-4) which may reside locally, or remotely. The characteristics of each of the databases is shown in Figure 2-5, and each of the above subobjects will be described below.

The status monitor provides either local or global status monitoring. The object's major service is to process the status reports of adjacent or subordinate nodes. This information is fed to a database containing the capabilities, condition and location of this asset (if it is local) or a group of subordinate nodes (if it is a global status monitor). The node then forwards it's status report to superior nodes.

The data correlator and discriminator is mainly associated with the sensor platforms. It works on raw sensor data to determine the properties of a detected object (multi-sensor correlation). The resulting properties are compared with those of previously detected objects in the threat database, and discrimination techniques are used to determine whether these 'new' objects can be identified with one of the previous objects.

The situation assessor can have local or global responsibilities. The local situation assessor works on locally stored object and status data to make inferences about the state of battle. The global assessor works on situation reports from subordinate nodes as well as any local data, in order to make inferences. These inferences are stored in the situation database for future reference.

The planner and controller may be thought of as two objects but the controller is strongly dependent on the information of the planner so here they will be considered one object. The planner/controller operates on incoming node orders and situation data in conjunction with some prestored plans. The planner then determines which countermeasures would be appropriate for the given threat situation, and supervises (controls) the subordinate nodes in executing these countermeasures. The planning and/or controlling responsibilities of an object will depend on where it falls in the control hierarchy - the lower an object is in the control hierarchy, the less the planning responsibilities it has.

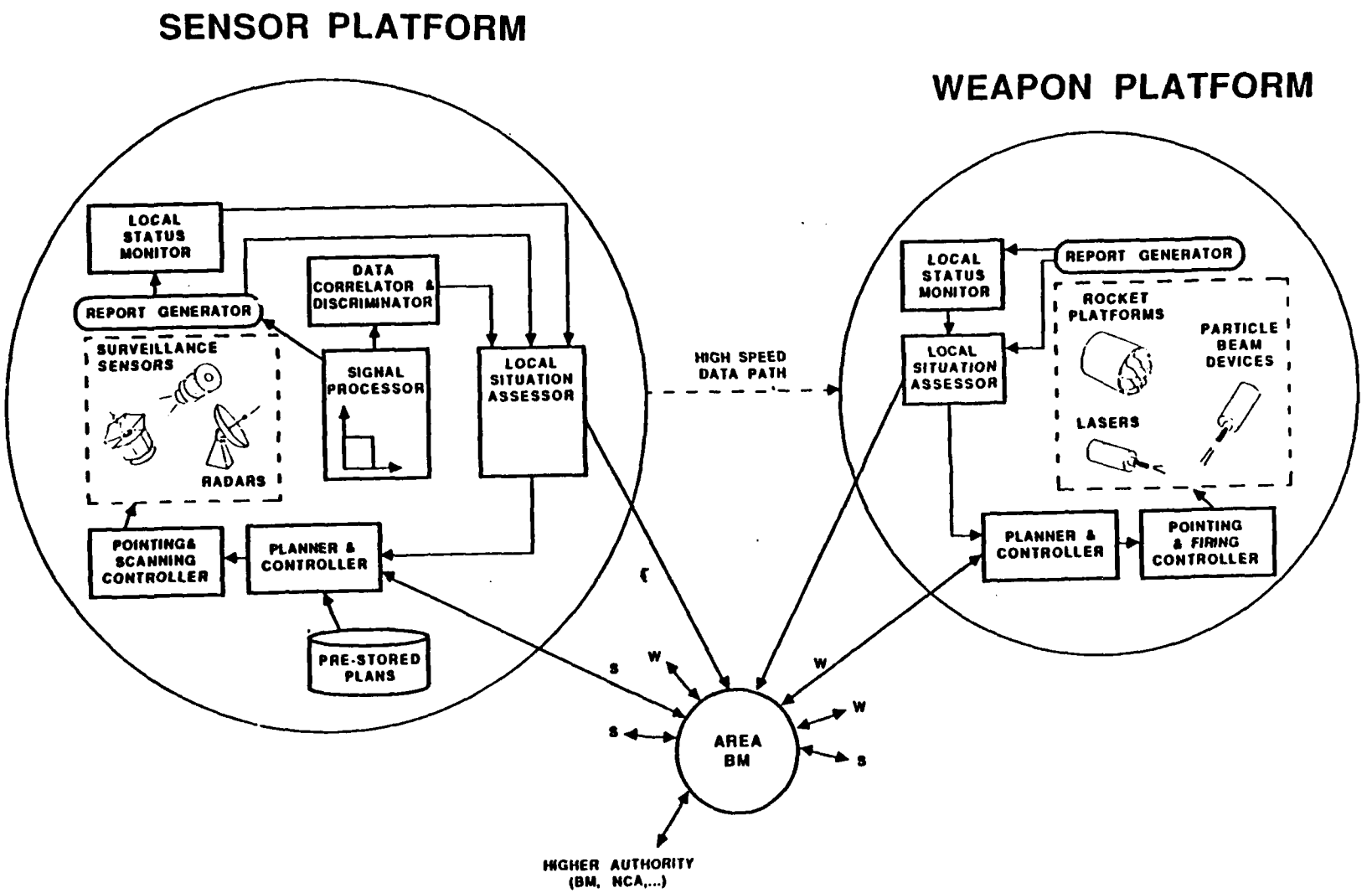


Figure 2-3 A Sensor vs. a Weapons Platform - Most objects will consist of different subobjects depending on their responsibilities.

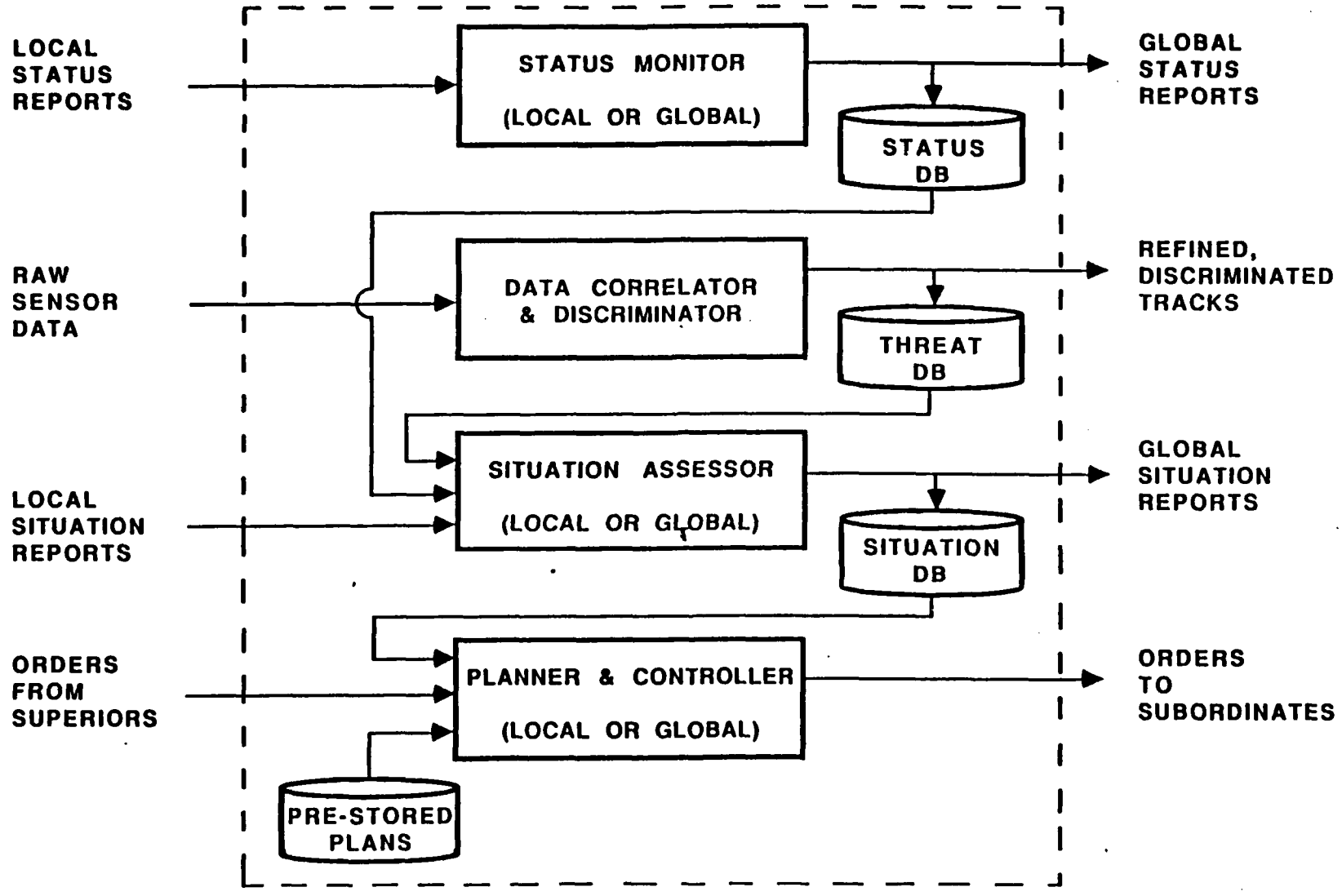
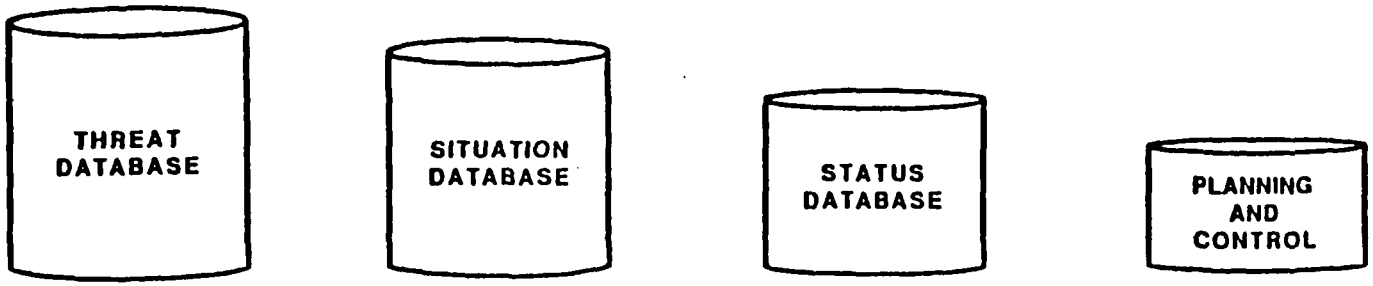


Figure 2-4 Generic BM/C3 Object Structure



CONTENTS	PROPERTIES OF ALL DETECTED OBJECTS (BIRTH-THRU-DEATH TRACKING)	ASSESSMENTS OF PERCEIVED THREATS (INFERENCES)	PROPERTIES OF SDS ASSETS	PRE-STORED PLANS OF LOCAL AND GLOBAL OPERATIONAL STRATEGIES AND TACTICS
	DYNAMICALLY UPDATED	DYNAMICALLY UPDATED	DYNAMICALLY UPDATED	NOT UPDATED DYNAMICALLY
TRANSACTION RATE	PROPORTIONAL TO RATE OF OBJECT CREATION DURING SALVO LAUNCH (≈ 1000/S)	FUNCTION OF INFERENCE MECHANISM FOR SITUATION ASSESSMENT	PROPORTIONAL TO *RATE OF REPORT* GENERATION ACROSS SDS	
SIZE	PROPORTIONAL TO NUMBER OF OBJECTS DETECTED, AND AMOUNT OF RAW SENSOR AND HISTORICAL OBJECT DATA	PROPORTIONAL TO NUMBER OF WARHEADS THAT PENETRATE DEFENSE	PROPORTIONAL TO NUMBER OF DISTINCT SDS ASSETS	DEPENDENT ON NUMBER OF PLANS

Figure 2-5 Database Characteristics

A high-level view of the system is depicted in Figure 2-6 where the BM/C3 objects are shown with comparable DOS configurations. The figure depicts the entire SDS consisting of a single DOS which integrates the "OS pieces" (operating systems that are local to each site or cluster), which are probably tailored to support unique requirements. These "OS pieces" are tied together via communications and a replicated kernel, forming the highly integrated distributed operating system (DOS). Realistically, and to simplify system design and development, the DOS may be considered as consisting of objects with their associated services. But, more than likely, the actual implementation of the DOS objects (beneath object/operation-oriented interfaces) might use more conventional techniques.

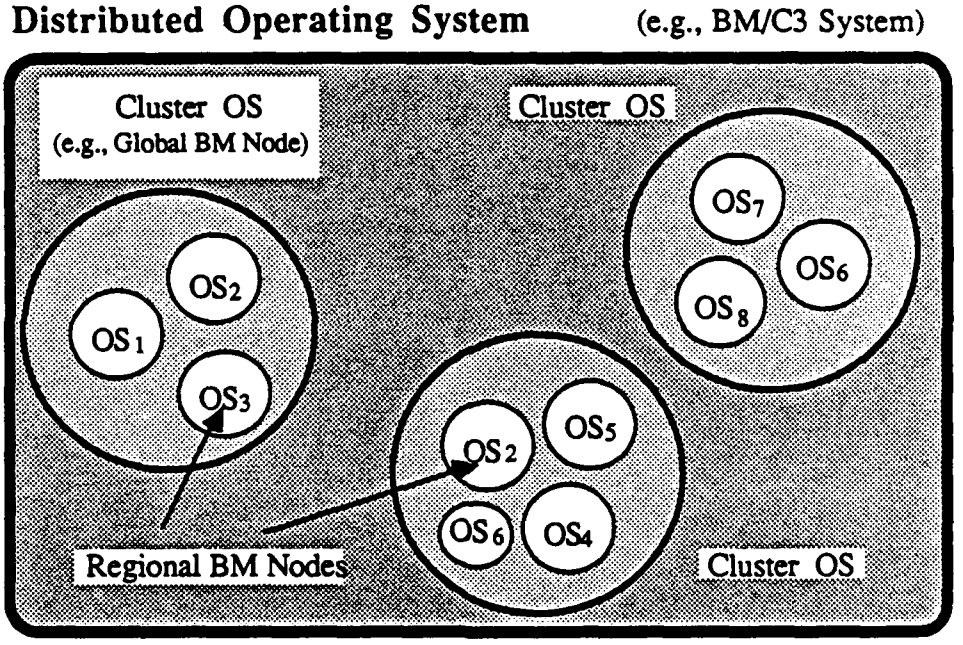


Figure 2-6 A BM/C3 DOS viewed as a single, highly integrated system of separate, tailored OS pieces

With all this in mind, what needs to be determined is the distributed operating system support necessary to achieve interoperability in a system such as the example system described above. Specifically, the key DOS functions which are necessary to achieve interoperability must be determined, and some design guidelines to promote interoperability must be set. In the sections that follow, a description of the core DOS components will be described along with the issues which must be addressed and some research recommendations.

SECTION 3. - SYSTEM ARCHITECTURE

This section describes the model, properties and semantics of a distributed, object-based system we recommend for a BM/C3 system. In addition to definite semantic recommendations, this section also discusses a few open questions regarding the model.

3.1 THE MODEL

The system paradigm chosen for this BM/C3 system is based on an object model. We chose this model because it supports the division principles discussed in Section 2.2 and because it is already widely used in DOS Research Projects.

This object model can be uniformly applied to all types of systems that will comprise the BM/C3 environment; i.e., from small to large systems, and both real-time and non-real-time systems. It can also be tailored for all BM/C3 applications through proper choice and programming of the individual objects.

Our object model consists of two basic abstractions, objects and threads. These components interface with each other through the mechanism of invocation to form a uniform and integrated system.

3.2. THE BASIC STRUCTURE

3.2.1 Objects

Viewed from the outside, an object is an instance of an abstract data type (or class) that has a specified behavior (methods). It inherits its behavior from its class. Classes can be arranged in an inheritance hierarchy, where new classes are formed by adding behavior patterns. This model ensures that applications are implemented using data abstraction (type specific interfaces) and information hiding (data accessibility only through the interface).

An object often has a memory of its own history (a state). This state data can be accessed and modified only through the operations defined by the object's class.

From the inside, the software realization of an object looks like a collection of private data (the state) and a group of procedures (methods) that can be invoked in response to input signals coming from other objects. The methods are executable code modules tied to the class description

3.2.2 Threads

The activity in the system is provided by threads. Threads are the objects that the schedulers in the DOS service. They have a state that represents the execution history of a computation. Threads are the carriers of computation and execute the code in other objects, using a sequential flow of control. A thread may span many objects. If the spanned objects reside on different sites, the thread not only crosses object boundaries, but also site boundaries.

Threads are implemented by processes. A process is an object that is scheduled for execution on a single processor. Objects may initiate threads of their own, e.g., to increase concurrency or to provide background activities such as garbage collection.

3.3 THE ENVIRONMENT

The hardware making up the distributed system is composed of sites connected by networks. Each site runs a portion of the distributed kernel that controls the functions localized to the site. The kernel provides direct support for objects, threads and invocations. The primary responsibility of a kernel is the delivery of messages sent between processes as operation invocations and responses.

The term cluster is used to refer to a collection of sites. A cluster generally corresponds to an installation of machines on one or more local area networks associated with an administrative organization (department, division, business, ship, etc.). The set of all clusters is called the multicluster environment.

Servers are objects that provide certain functions. For example, an authentication service might be provided to processes through the definition of a set of principal and group objects. A service is implemented as a set of processes executing on sites within a cluster.

3.4 PROPERTIES OF THE OBJECT MODEL

The following are the properties that object and threads have.

3.4.1 Objects

All objects in the system have the following properties:

Name

Every object is named. The names are unique throughout the multicluster environment and are used by the invocations to refer to the target objects. See the discussion on naming.

Location

Every object is located at one site. Some objects are replicated, meaning their copies are located at several different sites. Objects may be composed of references to other objects located at other sites. (See Figure 3-1.)

Entry Points

The operations defined in the object are visible to the outside world through entry points. In fact, the entry points are the only externally visible attribute of any object. Entry points must conform to a standard interface protocol that allows threads to enter and execute within objects.

Lifetime

The objects can be created and are explicitly deleted. In general, the lifetime of an object is long (compared to the lifetime of threads.)

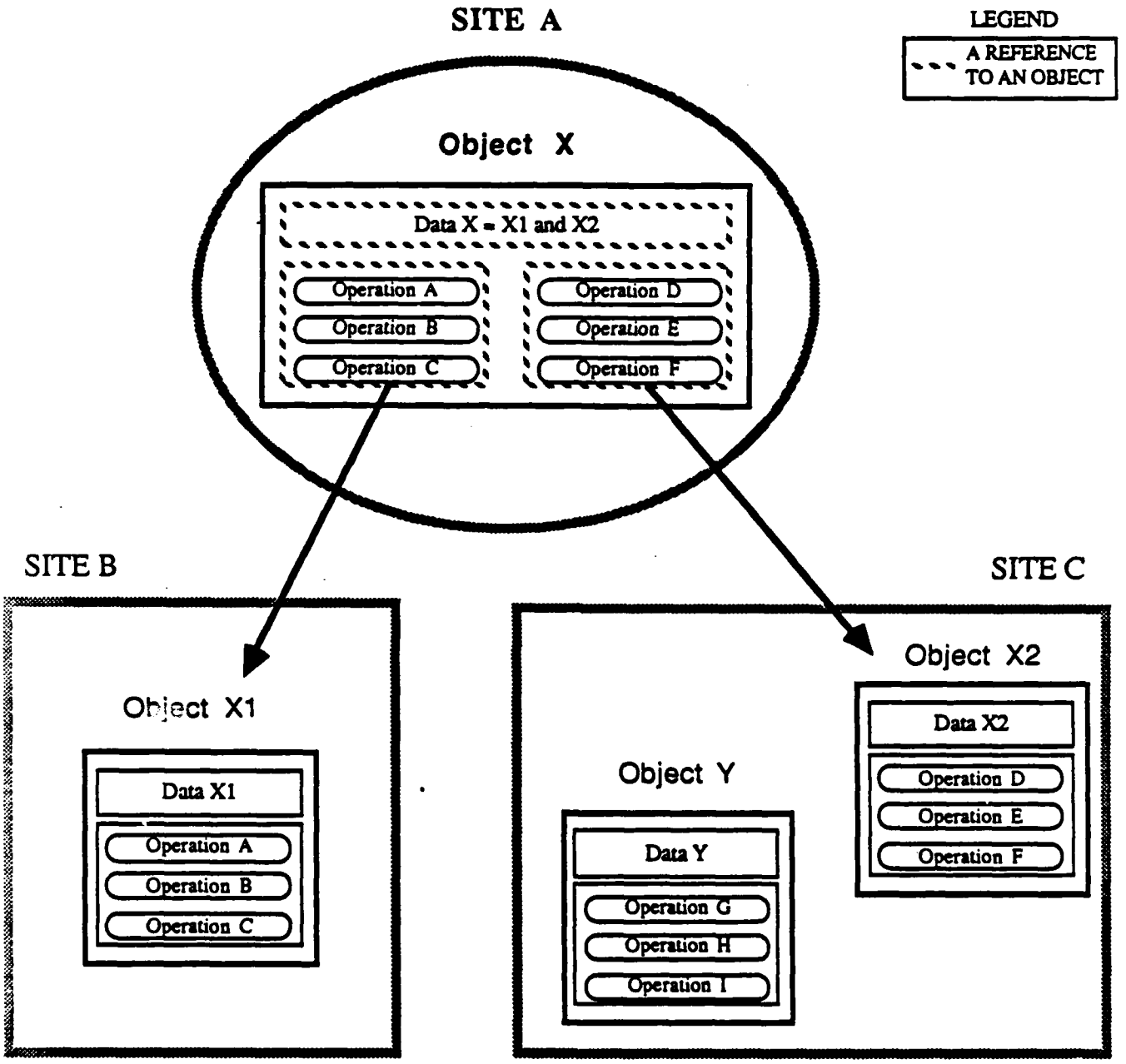


Figure 3-1 An Object Composed of References to Other Objects

Apart from these properties, some objects may have the following properties, depending upon the nature and application the object is programmed for:

Synchronization and Concurrency Control

Objects that allow (or expect) multiple threads to execute operations concurrently must provide synchronization methods to prevent race conditions on access to the shared, global data. Also objects that store data with consistency requirements must implement concurrency control schemes.

Recovery

Threads may update data in any object they enter. However, in the case where a thread aborts at some later stage, it may be be necessary to clean up the updates made by the thread. This is supported by objects having the recovery property, which can be used to restore data to some previous consistent state.

Replication

Objects may be replicated. Replication is necessary to maintain availability of data (and possibly processing) in case of failures (network failures as well as site failures) and is also needed for performance reasons. Those objects which need high availability may have this property; that is, they may exist on several sites. Recovery and concurrency control for replicated objects is more complicated than non-replicated objects because the state of a replicated object is distributed.

Real Time Response

Some (or all) operations in objects supporting real time control functions need to have real time deadlines specified. Objects involved in real time processing should have real time properties. Exactly what this means and how it is to be accomplished are hotly debated issues. This is further discussed in later sections of this report.

Access Control

Objects that need to be protected from unauthorized access have mechanisms preventing such access. See the sections on authentication and access control.

Security

In addition to access control (or protection), objects and kernels may implement some mandatory security policy.

3.4.2 Threads

The following are properties of threads:

Time

Threads have some time properties such as execution time, remaining time, deadlines, timeout requirements, and so on. These vary with the application and the operation currently under execution. In many cases, these time properties may be known only probabilistically.

Location

The thread originates at some location. As the thread visits new objects, it may spawn new threads, and may span machine boundaries.

Lifetime

A thread has a start time and an end time, the interval constituting its lifetime. The lifetime depends upon the nature and length of the computation done by the thread.

Atomicity

Threads may be atomic (or parts of a thread may be atomic). This denotes that the thread will either complete its function or have no effect; it will not leave things half done. Atomic threads are useful for consistency preservation as well as coordination amongst several operations.

Fault Tolerance

Threads may be replicated, or restartable, or may have backups that redo failed operations. Whatever the strategy, threads may be fault tolerant, making the computation being attempted by the thread resilient to faults.

3.5 SEMANTICS OF THE MECHANISMS

This object based system defines a set of semantic constraints upon the components. The semantics define the interaction of the components and the view of the system.

3.5.1 Invocations

Invocation is the mechanism that allows a thread to execute the operations defined in the objects. An invocation causes a thread to enter and execute in the object. The thread subsequently exits the object to either return or enter another object. Invocations can be nested.

Invocation may carry parameters. Invocation semantics are very similar to procedure call semantics. Parameters may be input parameters to the operation under invocation, as well as output parameters returned by the completion of the invocation.

Concurrent invocations of operations may have to be synchronized. The synchronization (or concurrency control) strategy used is programmed into the system using synchronization primitives.

Simple procedure call like semantics of invocations mandate synchronous invocations; that is, the invoker must wait for the invoked operation to return. This is often quite restrictive and may lead to low performance on some applications. Asynchronous invocations are supported in object systems to get around this problem when the need arises.

3.5.2 Transparency

Transparency provides cleaner semantics by allowing the system designer to view the interactions between system components conceptually without worrying about low-level details. Transparency in the system occurs at several levels. The naming system hides the locations of objects, leading to location transparency. The objects themselves hide their internal structure and operation implementations by defining external interfaces, giving rise to information hiding and functional transparency. Properties such as atomicity, fault tolerance, replication and so on are also effectively hidden by objects or threads giving rise to transparency in these areas.

3.5.3 Host Boundaries

The naming and addressing methods employed by the DOS kernels, make host boundaries invisible to the invoker of an operation. This makes the system more integrated and location independent than typical networks. However, the visibility of host boundaries are necessary for some functions, such as data configuration and maintenance.

Object creation and installation routines, global schedulers, real time controllers and so on need to see the site boundaries. This is achieved by having system management routines that allow specification of site names for specific purposes.

3.5.4 Access Control

Access control restricts various threads from accessing specific operations in certain objects. Threads have access privileges, and they can be disallowed from making some invocations. The ability of a thread to invoke an operation on an object depends on the access privileges of the thread as well as the object and the operation being invoked.

3.6 OPEN QUESTIONS

This section lists some topics about system architecture that are unresolved in the current state of the art. A short explanation of the problem, as currently workstood, follows each topic. The list is exemplary, not exhaustive.

3.6.1 Implementation

The implementation strategy and storage scheme for objects is open. Two major methods are the active object paradigm and the passive object paradigm. The implementation of threads in both these schemes is quite different, and each has advantages and disadvantages. The implications of the implementation strategy with respect to real-time constraints is another open issue.

3.6.2 Inheritance

As an instance of an abstract data type, an object is defined by the behavior of its class. Classes may be arranged in an inheritance hierachy or network. The semantics and implementation strategy (as well as the need inheritance) are unclear.

3.6.3 User Level Naming

User names are traditionally hierarchical. But hierarchical naming schemes need replication to avoid traffic and failure susceptibility of higher level directories. A well accepted scheme for object naming has not been reached. For more details see the section on naming.

3.6.4 Parameter Types

Parameters passed to objects have sometimes been limited to "value" parameters. Some other types may be copy, reference, segments and so on. The range of choices and feasibility (as well as importance) has not been adequately studied.

3.6.5 Binding

Binding is an open problem at several levels. The objects are named using user names at each invocation. The question is: should type checking be done at compile time (static binding) or run time (dynamic binding). Type binding of parameters is also an issue. Are conversions necessary, or should type checking be strong? This issue is further discussed in the section on addressing.

3.6.6 Common Operators

For reasons of programmability and better consistency, all objects should support some well defined common set of operations. What these operations should

be, and how uniformity across all objects is enforced, is open. In particular, what are the semantics for real-time operators?

3.6.7 User Interfaces

User interfaces to object systems should reflect the system structure of objects, threads and invocations, rather than the command languages of today which support files and programs. A consensus on how this can be achieved has not been reached.

3.6.8 Object Granularity

The size of objects can have an impact on the performance of a system. It is possible that different objects in the system may have different sizes, but what the granularity of these objects should be is an open issue. The effects that different sized objects have on the performance of the system needs to be investigated.

SECTION 4. - CORE COMPONENTS

A distributed operating system contains many components which cooperate together to ensure proper system operation. The DOSIRP panel has focused on nine DOS components they consider to be critical to a large BM/C3 system. By critical, it is meant that the nine core components represent those pieces of a DOS for which improper choice of protocols and mechanisms would severely impact system efficiency and the ability to meet real-time, fault tolerance, security and evolvability requirements and constraints.

The correct and efficient operation of any computing system can be directly related to the efficiency of it's communications system, it's resource allocation and scheduling algorithms, and the quality of it's timing and synchronization services. These components of a DOS have the most impact on the overall system performance and as such, the protocols and mechanisms must be chosen with utmost care.

The remaining DOS components reflect the DOS support needed throughout the lifetime of an object (i.e., creation, use, and destruction). At object creation a name, storage location, and a set of access rights/ privileges must be assigned to the new object. This involves the use of a naming, addressing, and access control service and a storage manager. During the lifetime of an object, it may invoke, or be invoked any number of times by other objects. The DOS support necessary to maintain system integrity includes authenticating the invoker of the object, and checking it's access privileges/rights before providing the requested services to the invoker. This necessitates DOS support for access control and authentication.

The nine core components (system-wide resource allocation, communications, timing, synchronization, naming, addressing, access control, authentication, and storage management) will be discussed in turn. Each of the sections will present guidelines for future R&D efforts, and the major issues pertaining to each of the components with respect to interoperability will be highlighted.

4.1 SYSTEM-WIDE RESOURCE MANAGEMENT (DISTRIBUTED EXECUTION/SCHEDULING)

This section discusses system-wide resource management. Four areas are highlighted: scheduling and resource allocation, process management, distributed execution, and fault tolerance. Resources are assumed to be modeled by objects. A thread, is assumed to model the active execution traces of programs. In particular, as a thread invokes an operation on an object, that operation may have a time constraint and criticalness properties associated with it. A process is then created to act as the mechanism used to implement the thread. This process is assigned time constraints and criticalness properties and this information is used by the scheduling algorithm. In addition, timing constraints and criticalness properties can also be associated with the entire thread itself, implying that a collection of process executions are subject to such constraints.

4.1.1 Scheduling and Resource Allocation

Good real-time scheduling algorithms form one of the most essential components of distributed operating systems underlying time critical applications. Without them it is difficult to predict how processes invoked dynamically interact with other processes, where blocking over resources will occur, and what the subsequent effect of this interaction and blocking is on the timing constraints of all the processes. Current real-time operating systems use priority based schedulers. This requires that actual timing constraints be mapped into priorities to meet these constraints - a time consuming, expensive and error prone process. Typically, one assigns priorities to processes and via extensive tests verifies whether this assignment results in processes and subsequently, threads, meeting their timing constraints. This, of course, only verifies that constraints are met under the tested conditions. Other conditions arise in practice, and a suitable algorithm must be flexible enough to handle such conditions. In dynamic systems such as BM/C3 applications, process priorities have to be changed according to the nature of processes in the system at any given time, and hence the problem of priority assignment and timing verification is further exacerbated.

New BM/C3 real time scheduling algorithms should be characterized as (1) decentralized - scheduling components at all (or at least multiple) nodes cooperate to schedule resources to achieve a system-wide objective, (2) dynamic - processes are scheduled as they are invoked, and (3) adaptive - the algorithms adapt to changes in the state of the system including overloads and failures of nodes.

The scheduling algorithm is at the heart of a real-time system and it should be driven by the timing, precedence, criticalness, and resource constraints of the processes and threads rather than by priorities which attempt to encode all of these things. The goal is to achieve transparent and uniform management of resources across node boundaries to optimize system-wide criteria. However, since scheduling is NP-hard, we must settle for scheduling algorithms which meet performance requirements at acceptable cost.

For explanatory purposes, we propose a decentralized scheduling facility (which can be defined as one or more abstract data types) composed of four modules (See Figure 4-1). The algorithms in this facility need to be able to handle nodes which are either uni-processors or multiprocessors. One module, a *dispatcher, simply removes the next process from a dynamically changing system* process table. A component like this will probably be found in all systems. However, even something as simple as the dispatcher is complicated in a real-time system. For example, consider a multiprocessor with processor/memory pairs on local buses and all sharing a global bus. There might exist a dispatcher in each memory or in global memory depending on many factors such as the bus contentions. This might subsequently require a different method for loading the dispatch table(s). Another module is the local scheduler which is responsible for organizing the dispatcher's table -- it implements the local policy and performs local resource allocations and process scheduling. A global scheduler is responsible for doing distributed resource allocation and scheduling, and approximating the achievement of a system-wide objective, e.g., having time critical threads meet their deadlines. Finally, a meta-level controller could adapt various parameters of any of the resource allocation algorithms, might even switch scheduling algorithms, and presents the user interface to scheduling and resource allocation policy. The meta-level controller may also act with fault tolerance mechanisms to guarantee high availability of services by providing some dynamic reconfiguration support and adaption of parameters to handle transient overloads and failure. It is too early to

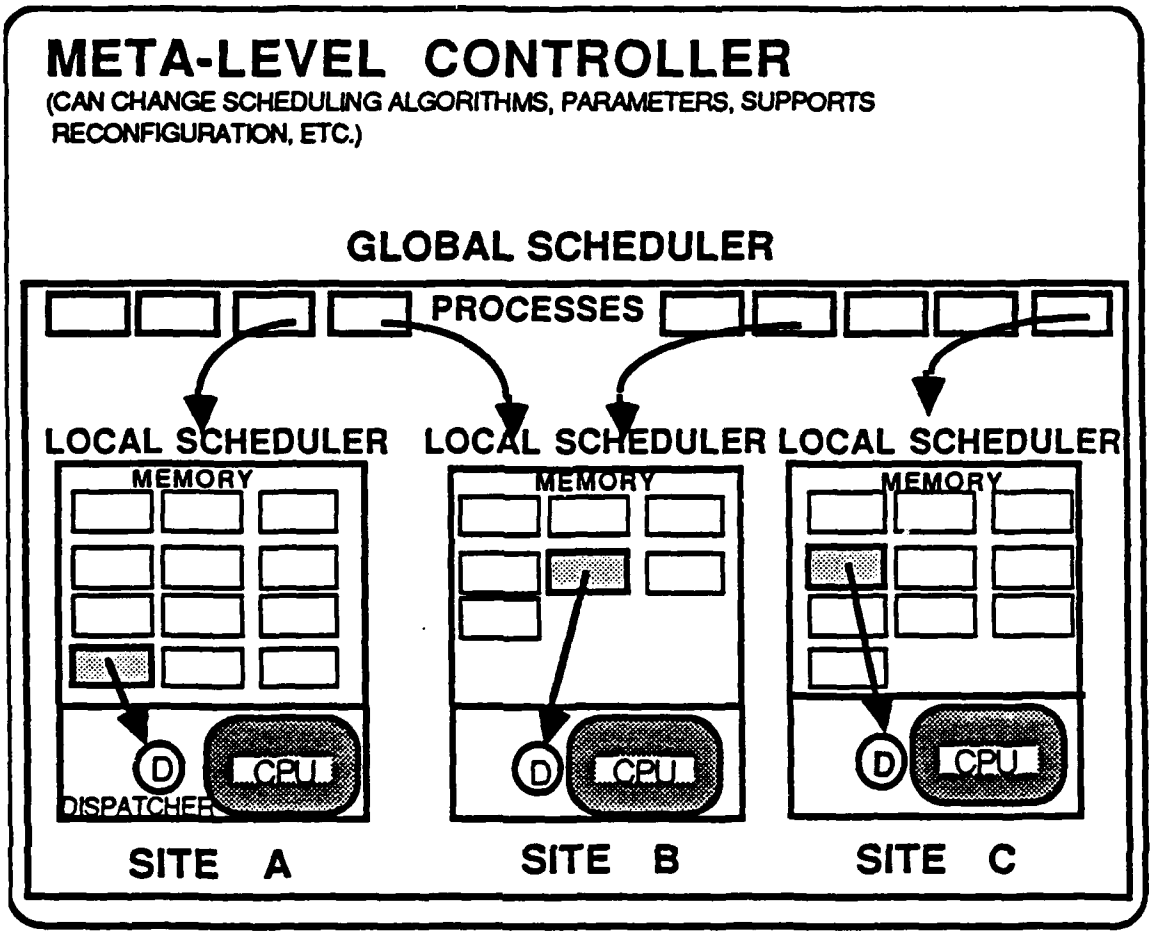


FIGURE 4-1 A Proposed Decentralized Scheduling Facility.

suggest interfaces to these modules, or how to divide them into one or more abstract data types. However, it is clear that the following information must be available to the scheduling modules for real-time processes and threads: worst case computation time of the process or a probability distribution function of the computation time of a process, timing constraints (such as deadline, ready time, period, etc.), resource requirements, precedence constraints, criticalness, preemptability, replication factor, and possibly security level. Other information might be optional. For example, if the distribution function of the computation time of a process is not available, then the average execution time of the process might be needed. The algorithms must be robust enough to handle preemptive and non-preemptive processes, as well as hard real-time and soft real-time processes.

It must be pointed out that the scheduling should be very efficient in the sense of making acceptable decisions with minimal delay and impact on application processes. A separate system processor may be warranted in many instances, because it does minimize delay and impact on application processes, and enhances modularity which has many subsequent benefits. In fact, it is not easy to separate the scheduling paradigm from the rest of the DOS design - they should operate synergistically. For example, the scheduling algorithms exist to help meet the timing constraints of the system, but if resource contention and or random requests cause tremendous uncertainty then it becomes next to impossible to analyze the timing properties of the system. The entire structure of the DOS must be suitable to timing analysis so that timing constraints can be verified to some appropriate degree. This may include programming the DOS in a style that caps the execution time of DOS primitives.

A key issue for resource management is the need to provide certain forms of predictability. However, more than lip service must be supplied. Predictability requires clean operating system primitives, some knowledge of the application, proper scheduling and resource allocation algorithms, and a viewpoint based on a team concept between the application and the DOS as well as within the DOS among its components. An important realistic and complicating factor is the need to be predictable in the presence of faults. This issue must receive more attention.

Conclusion: It is too early to recommend specific real-time scheduling and resource allocation algorithms, but guidelines for one approach is outlined above.

4.1.2 Process Management

The basic functions of process management are usually supported in the kernel. The kernel must be fast, support a fast context switch, respond to interrupts quickly, minimize intervals in which interrupts are disabled, etc. For example, to support process execution there might be a process control object and a context object. Organizing the information and physical resources of the system as objects has a beneficial effect on the management of these resources. In general, management schemes are simpler and easier to implement, use and modify. In addition, with the advent of multiprocessors the need for lightweight process

support has appeared. Such lightweight process support is also useful in real-time systems.

Examples of Process Management Functions needed include:

Create_process object - somewhat heavyweight, contains all those objects normally associated with processes

delete_process object - termination rules must be defined

create_lw_process object - ownership, sharing and communication rules must be defined

delete_lw_process object - termination rules must be defined

pause - more applicable for soft real-time processes although a pause primitive with a timeout T could be useful for hard real-time processes

resume - makes the process object ready again

delay_time - see pause primitive

replicate_process object - rules for replication, site assignment, coordination, etc. are required

assign_criticalness - an important characteristic of a processes: this value might change over time or as a function of load, mission phase, etc.

assign_security - trusted kernel might need to modify the security level

assign_level_ft - for fault tolerance purposes it might be necessary to assign a level of reliability which might dictate the number of copies required and/or style of ensuing fault tolerance

assign_timing_constraint - when a hard real-time processes (object) is created the timing requirements must be assigned; a standard specification language for this would be useful

query_process_state - since the system will be highly dynamic we envision adaption based on state values of the process (and also the system)

process_migration - this is facilitated by using the object model, for example, moving a process might entail moving all the objects it is currently accessing. Another possibility is to move the process object, context object, and the code object but let remote access exist for other objects needed by the process. Then the resource allocation algorithms can decide to perform remote access or move the object to the process based on estimated costs of remote access, etc.

The process_migration primitive supports the transfer of a process in execution. Two main issues are (1) deciding to move and to where, and (2) the mechanics of the move. The first question could be handled by the scheduling facility as described above. The second question is handled by standardized object structures and protocols supported by the process object management functions of the DOS.

4.1.3 Distributed Execution

Under the nomenclature of distributed execution and for purposes of discussion we consider three categories of distributed execution from simplest to more complicated. These categories do not necessarily cover all situations.

Category 1: Remote execution - A BM/C3 system at node i might make an initial assignment for a process object to a remote site for any number of reasons, or a user might specifically request such a remote execution via an object invocation. This is easily handled, and a number of remote job execution (RJE) protocols exist today. However, the metrics used to choose the site should take timing constraints into account which current RJE protocols do not.

Category 2: Process Migration for load balancing - Here processes in execution are migrated in order to balance the load in the system. This migration is significantly more difficult than RJE for several reasons. For example, the current state of the process must be accounted for, and it is not necessarily clear what "load balancing" should mean under various circumstances. Once a process has been created it has an address space, one or more stacks, some data structure such as a

process control block that describes its state, and a process identifier. For message passing systems physically moving a process typically requires the creation of a process control block (PCB) on the destination processor, copying the code of the process to the destination processor, allocating a stack on the destination processor and copying the current stack to it, and copying all system data structures related to the process such as file control blocks and message ports. These operations can be quite expensive. The problem with message passing systems with respect to process migration is that there is no easy way to translate the contents of different address spaces efficiently on the fly. Process migration in a shared virtual memory system is simpler because since processes share the same address space there is no need to actually move the control data structures nor regions of virtual memory. Pages will be moved as the migrated process uses them. It should be noted that process migration for continuous load balancing in real-time systems is not always desirable, but process migration is useful for fault tolerance and for temporary overloads. However, as in the Spring project, objects may be replicated at different sites so that only a signal need be transmitted when a load balancing decision is made, rather than actually migrating a task. This may increase the range of applicability of load balancing.

Category 3: Distributed programs - Distributed programs are a collection of physically dispersed concurrently executing programs cooperating to achieve some goal. The individual processes of the distributed program may have been started by RJE and/or experienced process migration. Typically, there is some level of concurrent execution in the distributed program depending on the application (e.g., types and amounts of IPC). Scheduling such programs subject to timing constraints has received very little attention to date. In fact, even without the complication of timing constraints it has been shown that if the scheduling algorithms are too simple then the system could actually experience a performance degradation rather than a gain by distributing processes. This is due to the interaction of IPC, context switching, and subnet delays.

Examples of issues related to distributed execution include:

a) supplying a process name service to provide transparency at the process level,

b) deciding the ownership and control issues for process hierarchies, distributed or not,

c) determining the policy on activating remote processes if they are invoked but not active,

d) providing support for replication,

e) providing support for RJE, load balancing, distributed programs, overloads, and reconfiguration, and

f) addressing migrations issues such as address space design, code movement, stack and other system data structure movement.

4.1.4 Fault Tolerance

The scheduling and resource management objects are replicated and act in a system-wide manner. Therefore, they provide a degree of fault tolerance. In addition, there are specific features that add to fault tolerance. These include the abilities to replicate processes, replicate objects, vote, coordinate via some protocol (e.g., a quorum based protocol), primary copy/backup options, transaction options, and any specific mechanisms such as supplied by the meta-level controller mentioned above, or some additional error reporting mechanism that could also be assigned to the meta level controller. For example, it might be important to report missed deadlines and other missed timing constraints which would feed back into the scheduling policy. One goal is to provide a protected execution environment. BM/C3 systems need integrated solutions to deal with real-time, fault tolerance, large system requirements, and security. For example, if a process's deadlines can be met only under restricted system states or configurations, reliability and performance may be compromised when the state changes due to overloads. Performance and predictability will have to be assessed under different levels of reliability.

4.1.5 Recommendations

- Real-time scheduling and resource allocation algorithms should be decentralized, dynamic, adaptive and have system-wide objectives.
- More research in scheduling approaches and algorithms that transparently manage resources across node boundaries is needed.
- The impact of time driven scheduling and resource management on DOS design and on object semantics should be studied.
- Various forms of, and approaches to, load balancing in real-time distributed systems need to be examined.
- Replicated objects and replicated execution time support is necessary.
- More research is needed into control strategies for scheduling (e.g., hierarchical, priority, etc.).

4.2 COMMUNICATIONS

Communications is the glue that connects objects of a distributed system. At the abstract data type level, communication between objects is achieved by object invocation. As described above in the section on system architecture, object invocation semantics for BM/C3 systems might include synchronous and asynchronous communication, as well as fault tolerance, security and real-time constraints derived from the application. Object invocation would be supported by a set of communication primitives such as RPC, datagrams, and transport level protocols, which, in turn, makes use of lower layers of communication protocols. In BM/C3 systems, heterogeneity demands that interoperability accommodate a variety of communication subsystems extending up through the network layer. Thus, it is primarily at the transport layer that the opportunity arises to affect interoperability through communication protocols. Since the transport layer is crucial to achieving interoperability, it is the focus of our interest in this section. As an aside we note that for efficiency reasons implementations of local object invocations usually bypass the layers of communication protocols. We will not

discuss such local implementation issues. Further, synchronization is one form of communication, but is usually considered at a higher level than as part of the communication protocols. See the section on synchronization for more details on this topic.

Considerable research and development work has been done in most areas of communication. In this section we concentrate on discussing just two major issues: the idea of communication facilities, and the transport layer protocols and mechanisms needed with respect to BM/C3 systems.

4.2.1 Communication Facilities

It is clear that communication must include RPC, datagrams, streams, real-time messages, secure messages, and fault tolerant messages. Real-time messages are those with time constraints such as periods or specific deadlines. Fault tolerant messages refers to those that have requirements to be received in spite of n failures. The specific techniques for achieving these types of communication is not clear. In fact, there is considerable debate on the form and content of such communication to support fault tolerance, security and especially real-time constraints. For example, some researchers advocate priorities as the mechanism to support real-time communication. Other researchers totally disagree, and have demonstrated situations where priorities are inadequate. Alternative solutions to priority are beginning to appear. Real-time communication might require specifying deadlines for messages and having algorithms that specifically use that information in sending messages. Because many real-time processes are periodic it may also be necessary to be able to dedicate bandwidth (maybe in the form of a special type of virtual circuit) for communicating processes such as allocating specific time slots on a bus. With regard to fault tolerance, mechanisms do exist for fault detection, isolation and recovery. However, what is needed is the integration of higher level concepts into the communications layers (e.g., protocols typically do not include the concept of threads).

To achieve communication, the socket facility of UNIX is one overall strategy. This facility supports various types of sockets (communication) including datagrams, streams, etc. There are uniform system calls to the socket facility. These uniform calls result in different types of communication depending on the type of

socket to which a process is connected. An advantage of the socket facility is that it can easily be extended. We want to make it clear that we are not advocating the UNIX socket facility itself, but are suggesting that the concept of developing a communication facility that is easily extended is correct. The socket facility, as implemented in UNIX, has been shown to be well suited for network operating systems in some non-BM/C3 environments, including software development and office automation. The set of primitives, system call interfaces, the communication types, etc. needed for a highly integrated DOS in a BM/C3 environment is currently unknown, and hence, using one or more easily extended communication facilities is probably a good choice. Each of these facilities may be built on a common set of low level mechanisms. The proper set of mechanisms for real-time communication is a current research topic.

4.2.2 Transport Layer

For good system performance, important higher-level operations must map efficiently to kernel primitives. In the case of the transport level and assuming extensive (if not primary) use of RPC, this rule suggests using a request-response protocol and kernel communication primitives that provide this facility, but we also believe that other forms will be necessary.

4.2.2.1 Problems with Current Standard Transport Protocols

Three main categories of deficiencies arise with most existing standard transport protocols of the genre as exemplified by TCP. We use TCP, a key current transport protocol, for comparison.

Poor RPC Performance

First, current protocols in the TCP class provide poor performance for remote procedure call (RPC) and network file access. This is attributable to three key causes:

- TCP requires excessive packets for RPC, especially for isolated calls. In particular, connection setup and clear generates too many packets.

- TCP is difficult to implement, speaking purely from the empirical experience over the last 10 years.

- TCP handles packet loss due to overruns poorly. We claim that overruns are the key source of packet loss in a high-performance RPC environment and, with the increasing performance of networks, will continue to be the key source. (Older machines and network interfaces cannot keep up with new machines and network interfaces. Also, low-end network interfaces for high-speed networks have limited receive buffering.)

Weak Naming

Second, current protocols provide inadequate naming of transport-level endpoints because the names are based on IP addresses. For example, a TCP endpoint is named by an Internet address and port identifier. Unfortunately, this makes the endpoint tied to a particular host interface, not specifically the process-level state associated with the transport-level endpoint. In particular, this form of naming causes problems for process migration, mobile hosts, multi-homed hosts, and replicated objects.

In addition, TCP provides no security and reliability guarantees on the dynamically allocated names. In particular, other than well-known ports, (host-addr, port-id)-tuples can change meaning on reboot following a crash.

Function Poor

TCP does not support multicast, real-time datagrams or security. In fact, it only supports pair-wise, long-term, streamed reliable interchanges. Yet, multicast is of growing importance and is being developed for the internetworking as well. Also, a datagram facility with the same naming, transmission and reception facilities of the normal transport level is a powerful asset for some types of real-time and parallel applications. Finally, security is a basic requirement in an increasing number of environments. We note that security is natural to implement at the transport level to provide end-to-end security (as opposed to (inter)network level security). Without security at the transport level, a transport level protocol cannot guarantee the standard transport level service definition in the presence of an

intruder. In particular, the intruder can interject packets or modify packets while updating the checksum, making mockery out of the transport-level claim of "reliable delivery".

Based on the experience gained by years of using TCP in a non-hostile, non real-time environment, a new protocol called, Versatile Message Transaction Protocol (VMTP), has been developed. The DOSIRP group considered this protocol during its discussions and, therefore, we discuss it below. Chesson's Protocol Engine is another protocol under development that should be studied for applicability to BM/C3 environments.

4.2.2.2 VMTP Overview

In general, VMTP is designed for the next generation of communication systems in non-BM/C3 environments. These communication systems are characterized as follows. RPC, page-level file access and other request-response behavior dominates. In addition, the communication substrate, both local and wide-area, provides high data rates, low error rates and relatively low delay. Finally, intelligent, high-performance network interfaces are common and in fact required to achieve performance that approximates the network capability. However, VMTP is also designed to function acceptably with existing networks and network interfaces. We discuss some of the details of VMTP to identify the main issues in developing a transport layer protocol, and to determine its feasibility for a BM/C3 environment.

VMTP is designed to support remote procedure call (RPC) and general transaction-oriented communication. By transaction-oriented communication, we mean that:

- Communication is request-response: A client sends a request for a service to a server, the request is processed, and the server responds. For example, a client may ask for the next page of a file as the service. The transaction is terminated by the server responding with the next page.

- A transaction is initiated as part of sending a request to a server and terminated by the server responding. There are no separate operations for setting up or terminating associations between clients and servers at the transport level.

- The server is free to discard communication state about a client between transactions without causing incorrect behavior or failures.

The term message transaction (or transaction) is used to mean a request-response exchange in the sense described above.

VMTP handles the error detection, retransmission, duplicate suppression and, optionally, security required for transport-level end-to-end reliability.

The protocol is designed to provide a range of behaviors within the transaction model, including:

- Minimal two packet exchanges for short, simple transactions.

- Streaming of multi-packet requests and responses for efficient data transfer.

- Datagram and multicast communication as an extension of the transaction model.

Example Uses

- Page-level file access - VMTP is intended as the transport level for file access, allowing simple, efficient operation on a local network. In particular, VMTP is appropriate for use by diskless workstations accessing shared network file servers. Is it also useful for highly interoperable BM/C3 environments? This question is unresolved at this time.

- Distributed programming - VMTP is intended to provide an efficient transport level protocol for remote procedure call implementations, distributed object-oriented systems plus message-based systems that conform to the request-response model.

- Multicast communication with groups of servers to: locate a specific object within the group, update a replicated object, synchronize the commitment of a distributed transaction, etc. VMTP supports unreliable multicast. It seems necessary to also support reliable multicast for BM/C3 systems.

- A very limited degree of distributed real-time control is supported with prioritized message handling, including datagrams, multicast and asynchronous calls.

Overall, VMTP provides an efficient (with respect to TCP), reliable, optionally secure transport service in the message transaction or request-response model with the following features:

- Host address-independent naming with provision for multiple forms of names for endpoints as well as associated (security) principals.

- Selective retransmission.

- Rate-based flow control to reduce overrun.

- Idempotent message transactions with minimal server overhead and state.

- Secure message transactions with provision for a variety of encryption schemes.

- Multicast message transactions with multiple response messages per request message.

- Datagram request message transactions with no response.

- Priority processing of transactions, conditional delivery and preemptive handling of requests as partial support for certain forms of real-time communication.

- Forwarded message transactions as an optimization for certain forms of nested remote procedure calls or message transactions.

- Multiple outstanding (asynchronous) message transactions per client.

- Multi-packet request and response messages, with a maximum size of up to 4 megabytes per message.

- Simple subset implementation for simple clients and simple servers.

A question hotly debated by the DOS panel was whether VMTP is sufficiently close to an appropriate protocol for BM/C3 systems that it should be either adopted or at least studied with respect to the special requirements of BM/C3 systems, or whether it is largely inappropriate primarily because it was designed for a totally different environment where fault tolerance and real-time constraints are not as severe. There was no consensus on this issue. In developing VMTP, a multitude of design decisions were made in an integrated manner to achieve a complete protocol. Therefore, VMTP can serve as a model that identifies both the main issues and the specific choices made by its designers for those issues. Revisiting each of those issues for BM/C3 requirements (in detail) could and should be done.

4.2.3 Recommendations

- Encourage the development of one or more real-time transport protocols and compare them under different conditions, for example, under overload conditions.

- Get BM/C3 DOS designers to document their requirements and rationale for transport protocols.

- Perform research on transport layer mechanisms which would support a wide range of BM/C3 transport layer policies.

- Investigate the development of a communication facility for BM/C3 and all that it implies, e.g., what is the proper set of message types needed to be supported, how can those types evolve, what are the communication interfaces and primitives, etc.

- Study the applicability of VMTP in a BM/C3 environment concentrating on real-time scheduling and performance, fault tolerance, security, and adaptability.

4.3 TIMING SERVICES

The time service in a distributed system should provide current time of day that is accurate (to some requirement) as well as consistent across the nodes of the distributed system. That is, the difference in current time provided by any two nodes, asked simultaneously, should be less than some tolerance, epsilon. An important question to be answered is the impact on the system design, operation and maintenance if a consistent time cannot be guaranteed. It is likely that time across a WAN is not synchronized to a high degree.

4.3.1 Time Service Interface

Examples of standard general time functions could include:

Time Functions

Delay(seconds, clicks) Suspend the execution of the invoking task for the specified number of seconds and clicks where a click is one microsecond. (The accuracy of Delay reflects the machine clock resolution.) The task may be unblocked by Wakeup.

GetTime(time, clicks) Return the current time in seconds and clicks.

SetTime(seconds, clicks) Set the kernel-maintained time to that specified by seconds and clicks.

Wakeup(taskid) Unblock the specified task if it is delaying using Delay.

The kernel provides operations for reading the time, setting the time, delaying for a time period, and unblocking a delayed task. Time is maintained in SYSTEM.CLICKS units, which should be as fine-grain and accurate as practical. Other time functions might be necessary for real-time control such as RESYNC(clocks) and QUERY(time_remaining).

4.3.2 Time Synchronization Protocols

Clock synchronization is the principal example of approximate agreement. Synchronization of clocks can apply to logical or real time. Clock synchronization involves two problems:

1. Join: synchronization of a new clock to run with a group of clocks.

2. Resynchronization: periodic correction to keep a group of clocks closely synchronized, assuming "most" are fairly close.

There are three general approaches:

1. Synchronization wave:
2. Averaging:
3. Multicast synchronization with averaging arguments:

Approaches 1 and 2 are expensive and incur substantial simultaneous traffic. They entail each node sending a flood of messages to all other nodes at the same time as others, namely at the beginning of a round. This behavior is disastrous for the performance of a contention network, given the orchestrated collisions that result. In addition, in a BM/C3 environment, this flood of "overhead" traffic possesses a threat to critical communication unless there are appropriate guards in place. A mechanism to prevent temporary load bursts from triggering the recovery mechanisms is needed.

Approach 3 basically combines the two approaches with asynchronous "rounds" between different nodes, thereby avoiding the massive collision problems. It has good behavior in experimental testing but has not been tested under worst case conditions.

The choice of fault tolerance mechanisms will ultimately depend on the application requirements and the system configuration. One serious problem for BM/C3 systems is that none of the fault tolerance mechanisms currently mentioned in the literature takes into account real-time constraints or tightly synchronized clocks.

4.3.3 Recommendations

- Further research is required to bridge the gap between theory and practice in time synchronization.
- Hardware support for clock synchronization on a LAN might prove beneficial and should be investigated
- It is necessary to study the impact of having loose, or even no time synchronization in BM/C3 systems.

4.4 SYNCHRONIZATION

Synchronization techniques are used to protect critical data or resources from simultaneous use, or to order execution. Common synchronization techniques include semaphores, conditional semaphores, monitors, events and signals, 2-phase locking, optimistic concurrency control, and various forms of IPC primitives. Such synchronization primitives are well known and used extensively in OSs. Are these primitives fault tolerant and tailored to real-time? In general, the answer is no. Special primitives may be required, e.g., a timed semaphore which releases a process if the process is blocked on a resource for time T, and dequeuing a semaphore based on time constraints rather than first-come-first-served or simple priority.

Rather than discuss these common and well understood synchronization techniques, we discuss synchronization from another viewpoint, i.e., as a form of agreement protocol. It is possible to consider synchronization and time (See Section 4.3) services under the general heading of agreement mechanisms and protocols. That is, distributed modules may have to agree on a value, or on-going process, either exactly or approximately. For example, in an atomic transaction all participants must agree whether to commit or not. With clock synchronization, tasks must agree on time (more or less). More generally, in distributed process control, tasks must agree on the state of the process and the control action to take. In applications in general, there are consistency constraints between update units that need to be maintained, reflecting another form of agreement.

Various services and algorithms make a range of assumptions in their implementation, including properties of; (1) participants, (2) communication facilities and (3) synchronization. Typical assumptions cover:

- Perfect communication vs. message loss, delay, modification.
- Perfect participants vs. fail-halt vs. "Byzantine" failure (behaves maliciously after failure).
- Synchronous rounds vs. asynchronous.

There are different protocols, mechanisms and costs corresponding to the different assumptions. We cover a few cases here. However, very little current work has considered real-time constraints.

4.4.1 Exact Agreement

Simple examples are 1-phase and 2-phase commit protocols.

The problems with 1-phase commit include:

- A participant may not (be able to) agree, may crash.
- Coordinator may crash after sending decision to a subset of participants.
- Participants do not know whether everyone agreed or not.

Also, a coordinator (dictator) is assumed.

The more common solution is the well-known 2-phase commit (agreement) protocol.

Several problems arise with 2-phase commit:

- If the coordinator crashes before a participant receives a commit or abort (but after agreeing to commit), it must wait to learn the coordinator's decision. (so-called "blocking" commit protocol)

- There has to be a "trusted" coordinator, i.e. suppose coordinator sent different decisions to different participants?

- The solutions to coordinator crashes and blocking include so-called non-blocking (3-phase) commit protocols. The problem with non-blocking protocols is that they are more costly in messages or delay (or both) than 2-phase and blocking rarely happens, especially with replicated coordinator information. Again, in all these protocols, no special consideration of real-time constraints have been made. We hypothesize the need for real-time transactions in which the concurrency control and commit protocols are tailored to real time systems.

There seems considerable potential for standardization of 2-phase commitment protocols in non-time critical environments. In fact, ISO already has a draft, called CCR which is largely based on the IBM LU 6.2 specification. However, this protocol assumes that there is great importance in avoiding logging of transaction records (commit/abort information) and introduces an extra message per transaction and participant, basically to acknowledge a transaction commit or abort. In general, this protocol is oriented to large-scale mainframe nodes more than workstation or BM/C3 applications.

Another alternative is the class of transaction protocols in which the transaction protocol is primarily for coordinating between peer transaction managers at different nodes, each assumed to perform its own logging and (sub) commit interaction with local participants. Again, this work has not addressed real-time systems so its applicability is questionable.

A third scheme assumes a relatively "stateless" client which directs the transaction, with stable storage for transaction records modularized in a separate transaction log server. This protocol also makes use of multicast to efficiently initiate prepare-to-commit, commit and abort. The long-term storage of commit decisions eliminates the extra acknowledgement message in CCR. The direct

coordination of the transaction and 2-phase commit by the client eliminates the need for per-node transaction managers.

The above discussion points out a source of significant variation; namely, the assumptions that are made about the nodes and the relevance of logging transaction records. Note that this variation arises without even considering differences in fault-tolerance, real-time, and security requirements.

Atomic broadcast has been proposed as a more general exact agreement mechanism. With atomic broadcast, a message can be sent to a group such that either all nodes receive it reliably or no nodes receive the value. These protocols entail a trade-off between number of messages versus the response time to when the broadcast is known to have completed (or is known that it will eventually commit). In a BM/C3 environment, it appears questionable to use these types of protocols except for background agreement activity. An additional problem is introduced by the all or nothing property of atomicity. In a BM/C3 environment, it may be unacceptable to have the result be "nothing".

An additional problem arises if the coordinator sends out different values to different participants. Can all correct participants be sure to agree on single value even if there are incorrect participants or coordinator? This problem has been studied as the Byzantine agreement problem.

4.4.2 Byzantine Agreement

The Byzantine Agreement problem is commonly phrased in fanciful terms of the Byzantine Generals problem: can the generals decide when to attack (or not to attack) so all non-traitors do same thing at same time, even if some of the generals are traitors? One can show it is impossible with more than t traitors, for some value of t as a function of the total number of generals.

Solutions usually assume: (1) perfect communication, (2) exact synchronization and (3) authentication. In theory, (1) can be addressed by mapping communication failures onto processor failures. To avoid communication failures becoming a problem in practice (such as in a BM/C3 environment under stress), one can employ standard reliable communication techniques (such as acknowledgement, timeout and retransmission and possibly even forward error

correction) to reduce the probability of transmission errors and interference resulting in an "observable" communication failure. (The success of this approach is intertwined with the feasibility of maintaining any level of communication under stress.) To handle (2), one can introduce separate clock synchronization for phases or use logical clocks. Given the need for synchronized clocks in the BM/C3 environment, assuming reasonable clock synchronization is realistic unless the rounds or phases must occur very rapidly. Assumption (3) is easy to handle with encryption techniques.

The following subsections capture a few additional comments on fault-tolerance and security relative to exact agreement methods.

4.4.3 Fault Tolerance

Two-phase commit suffers from fragility in the sense that all participants must agree-to-commit or else the agreement fails (defaults to abort). Thus, a strict implementation is very fault-intolerant. One solution is to abort the transaction and retry the transaction with a new (or reduced) set of participants, trying again to commit the new transaction. While feasible in some cases, this approach may well induce a "timing fault" in a BM/C3 system if the action has to be redone (not forgetting the cost in some cases of undoing subactions associated with the aborted action). A third alternative is to use action-specific judgment to revise the list of participants such that the action has satisfactory semantics. For example, one might make the judgment that the responding participants to a commit to engagement is sufficient to commit to an offense action.

The concurrency control aspect of support for transactions is the feature where almost all current systems have chosen a lock based scheme; however, we have not seen a locking algorithm for nested transactions that takes into account deadlines and criticalness. One approach which begins to address some of the performance questions involves support for non-serializable execution of transactions by allowing an object designer to specify operation compatibility based on semantic properties of operations.

While locking is the most widely accepted concurrency control method, it remains to be seen whether the nature of BM/C3 systems (i.e., the changing

requirements from peacetime to battle mode, as well as provisions for degraded performance modes) would favor optimistic or time stamp ordering methods.

Another issue regarding fault tolerance is the ability to detect failures. In a BM/C3 environment, failure detection should be done through a combination of time-outs and agreements, which will eventually accommodate replication mechanisms. An issue which is receiving considerable debate concerns when a site is considered a failed site.

4.4.4 Security

A base level of security is provided in agreement protocols using encrypted communication that guarantees, up to the limit of safety of the encryption and key distribution mechanism, that messages received are unmodified and were sent by the identified principal, and not a replay. If this security system is not compromised, the coordinators and participants can proceed as though there is no security threat.

If the security system is assumed breachable, there needs to be further compartmentalization of the key distribution and encryption mechanisms for "damage control". In particular, one would require that compromise of one key system would not immediately compromise all communication, only some. This security structure would be required by the system as a whole. However, one can look to Byzantine agreement techniques to aid in detecting security breaches. In particular, the detection of an inconsistent coordinator as suggested above would be grounds to suspect a security violation (the other explanation being a truly faulty node whose problems were not being caught by its error detection mechanisms).

4.4.5 Recommendations

- There seems to be a basis on which to explore standardization of an atomic transaction protocol in a non real-time environment. The exploration should start with a narrowing of the three alternatives discussed above.

- It is an open question whether atomic transaction protocols can be made suitable for a real-time environment and this needs to be investigated.

- Determine whether synchronization techniques in a real-time environment should be based on various criteria suitable to meeting timing constraints, e.g., the work on non-serializable transactions seems to have promise.

- The Byzantine agreement problem has been sufficiently explored in the general literature that some standardization in this area might be possible. However, research is required into the real costs, trade-offs and applicability of these protocols, which appear too expensive and rigid for many BM/C3 environments and uses.

- There should be research into adaptive protocols, e.g., commit protocols that switch from logging abort decisions (presumed commit) to logging commit decisions as the environment changes.

- Failure detection should be done through a combination of time-outs and agreements.

- An issue which must be resolved concerns when a site is considered a failed site.

4.5 NAMING

Naming applies to a wide variety of entities in a DOS. Those entities in an object-oriented, wide-area DOS which must clearly be named are objects, types, threads, hosts, and clusters.

A DOS commonly supports at least two levels of names: system names and user names. System names are used by the DOS kernel to uniquely identify and locate objects. User names are generated by applications and users to provide human usable names for all entities in the system. The system and user name spaces are used entirely differently, and therefore have different properties with respect to their generation and maintenance. The issues that need to be addressed when considering each level of name space for each entity (or object) are:

- name generation/administration:

  -what is being named,

  -how are names generated/allocated to avoid conflicts,

  -in what context are names unique,

  -what information is contained in them,

  -what is the minimal scale of each field;

- use:

  -how are they used in referring to objects,

  -how is name translation between spaces done,

  -how are names used for addressing entities,

  -what level of software uses each name,

  -what names are used in structured objects, and

  -how do names impact object migration?

- name-space maintenance:

  -are names immutable,

  -how are names changed and removed,

  -how are names deleted when objects are deleted,

  -can names be reused;

- replication support:

  -how are replicated objects named?

Naming is critical in the DOS because it is used by all layers of software. An example of its importance is the treatment of replicated objects. Generally, data replication is treated as a resource management issue that is built on top of the session layers of the DOS. However, since addressing is a key session layer function, the choice of replication naming has an impact on this layer.

The propriety of a naming convention largely depends on the difficulty of administering the space. Globally unique names greatly simplify the use of names, but can complicate their generation (e.g., TCP port allocation). A most common approach to selecting unique names is appending a context to the name which distinguishes the selected name from duplicates elsewhere in the multicluster environment, where a cluster corresponds to a collection of hosts at an installation. This approach scales only if the context itself is defined recursively by context (e.g., hierarchical Internet domain names host.org1.org2.... orgn.orgtype), or the context and name fields are wide enough to provide coverage over the name space.

4.5.1 System-Level Names

4.5.1.1 Names of Objects

The system-level name of an object must be globally unique to ensure its accessibility from any context. Experience has demonstrated that object names which contain temporary information (e.g., location hint) become difficult to use because system-level names for objects persist in other objects and become difficult to update. Experience has demonstrated that system-names should be immutable in systems where object identifiers are allowed to be embedded in other objects. This type of reference, common in most object oriented systems, makes it virtually impossible to know when a system identifier can be changed without leading to outdated identifiers. Making system- level names immutable simply requires choosing names from a very large space (e.g., approximately 100 bits for the entire Internet). Renaming of objects allows names to be smaller in size, but also requires limitations of the use of names to avoid conflicts.

A replicated object can be viewed as a set of objects each with different names, or as a single abstraction which may have copies on different hosts. Giving an object's copies different names either limits the use of whatever copy is available (because you have the wrong name), requires that you have multiple names for the same object, or leads to maintaining a list of all the copies at the application layer. An alternative approach is to treat an object's system-name

as the unique identifier for a data abstraction, where that abstraction may have that property of being located in different places.

An object's creation should not depend on the availability of any remote host in the system. And since object creation is a frequent event, the kernel on a host should be able to generate system-level object identifiers. The simplest structure for an object is the triple (type, host, uno), where type is the object type of the object, host is the machine where the object is created, and uno is a unique identifier for the object on the host. Local identifiers should be unique throughout the lifetime of a host to ensure that object names are never reusable.

This naming scheme makes the assumption that hosts and types are themselves globally unique. And because of the nesting of names, it dictates that host and type system-level names are defined in different name spaces than objects (although they may have object name aliases). Host names need to be used by the transport layer protocol. It seems reasonable to adopt the name for hosts based on the protocol being used. This also has the advantage of drawing on existing host name distribution and authentication mechanisms. If more than one transport layer is being used, one format should be selected and a mapping function defined in other contexts.

4.5.1.2 Names of Types and Clusters

The naming conventions adopted for types and clusters are dependent on the nature of interactions between clusters. If the interaction is limited, global naming schemes are unnecessary. However, the limited history of distributed operating systems has shown extensive evidence that interoperability between clusters is an important property. We next consider global naming of types and clusters in the multicluster environment.

In a single cluster system, a type has only one interface, implementation, object set, etc. But in the multicluster environment, types need to be interpreted with respect to a context. A 'type' may have a different implementation and set of objects in each cluster. Since hard interfaces are the critical property underlying a DOS, a type should identify an interface. A type may then be instantiated in different clusters, possibly with different implementations and objects but always with the same interface. This allows the

same interface to be adopted regardless of the properties of the type in different clusters (e.g., fault tolerance, security, etc.). This property of object based systems helps support heterogeneity, modifiability, transparency, and adaptability.

In order to maintain consistent interfaces and support call compatibility between clusters, a type should have the same system-level name, regardless of where it is instantiated. Since the type name space can be large and types in a development cluster will be created fairly frequently, the 'context' naming approach facilitates system-wide type name uniqueness without administration. A type could be named by: a genesis cluster indicating where the type is first created, and a local identifier, unique within that cluster. This approach makes it possible for service peers (i.e., server processes of a type executing in different clusters) to communicate without mapping between type numbers. The approach also can be used to extend the domain of a type's instantiation, i.e., the set of hosts where the objects defined by a type may reside, beyond the boundaries of a cluster.

The advantage of the 'context' naming approach is that type generation and distribution can be entirely automated. The disadvantage of the approach is the required size of the resulting type name space. This is particularly costly if type identifiers are used as components of object's system-level names. The name space can be reduced by manually administering the allocation of new types.

The same tradeoffs that apply to type naming apply to the naming of clusters. Clusters are created much less frequently than objects or types. It seems reasonable to administer the system-level name space of clusters in a centralized fashion, with backup. And since the name space of clusters is relatively small, it seems reasonable to use a flat space.

4.5.2 User-Level Names

User-level names are generated by the user, and need to be organized in a user understandable way. The information in a user-name is assumed to be at the discretion of the user, and, since users are fickle with names, is assumed to change occasionally. Since user-level names may change at irregular intervals

they should not generally be placed in objects; otherwise, out of date names will occur frequently.

The most straightforward naming scheme is to name objects by the host they reside on, as adopted by Apollo's Domain system. This approach fails at the system level because objects' system- level names must be immutable, and cannot be if their names contain host addresses and they migrate between hosts. Naming by hosts at the user-level violates our assumption of host and network transparency.

A hierarchically organized, logically centralized catalog for maintaining symbolic names for all types is a natural extension of conventional file system naming. Hierarchical naming strategies also correspond to military command structure. The Catalog Service would maintain objects of type directory, which contain a list of entries for object names. The user interface becomes responsible for translating user-level names into system-level names; user-level names would generally only be used within the user interface.

This approach has several strengths and weaknesses. The strengths are the *convenient user services, including the ability to support:*

- multiple user-level names for each object,
- symbolic links,
- extensibility of the namespace, by allowing lookups in other types which implement directory operations; this allows, for example, linking to remote directories (in other clusters) or linking to parallel names spaces maintained by application managers,
- multiple (alternative, evolving) user-level symbolic naming systems based on same system-level names,
- caching of both system level (e.g., location hints) and application level (entirely dependent on the application) information in the directory entries,

•standardized use of system-level names at all levels below the user interface,

•independent development and evolution of name services, separate from other system services and applications

The accessibility of an object by a user is dependent on the ability to translate between the object's user-level name and its system-level name. Therefore, the design of the name service facility must take into account the consistency of the data and its availability. Since a great many objects will have user-level names, the catalog must also be scalable.

The chief problem with a multi-level naming structure is the separation of the objects from their name space. Ensuring that all of the user-level names for an object are removed when the object is removed requires either maintaining a link from each object to its user-level names, or having the service that manages an object also manage its user-level names. Given the number of objects expected in a BM/C3 environment, this may be prohibitively expensive. Another byproduct of the separation is that the properties of objects (e.g., consistency, availability, etc.) are independent of the properties of their user-level names. This independence may not be desired in many cases.

One alternative to the name server previous described is the approach taken in Grapevine [Birrell et al. 82]. Grapevine is a naming facility intended primarily for electronic mail delivery. In addition to supporting name service distribution and replication, it also handles authentication, access control, resource location, and delivery. The name space is based on individuals and groups, where an individual is analogous to an object name and a group is analogous to a directory. Naming provided by Grapevine can be nonhierarchical, which is more flexible than a strict hierarchy but can be more confusing for users.

Grapevine provides both mapping between names spaces and mapping between a name and an entity's location (location binding). It is also possible to have a name server which simply provides name translation, and have the DOS kernel provide location binding and message delivery services. The advantage of combining name mapping and location binding into the same functional component of the DOS is the simplicity, uniformity, and performance of the

mechanisms. The advantages of separating them are that name spaces are more easily scalable if separated from lower layer delivery services, and there is more flexibility in being able to use different naming or binding schemes.

The Apollo Domain system prepends symbolic host addresses to symbolic names. This approach greatly simplifies name translation and improves performances, but it fails to provide network transparency and makes transparent object replication difficult.

Another alternative to a logically centralized naming server is to move names into managers of objects. The manager of a set of objects becomes responsible for managing the names of the objects as well. This decentralized naming strategy may be appropriate in applications where objects represent physical names (people, hosts, cities, ships, etc.), but does not lend itself toward a globally consistent, type-spanning naming system.

Hybrid solutions, such as service-maintained namespaces accessible through the name server facility, combine many of the advantages of both approaches. Hybrid approaches complicate internal mechanisms by creating an externally unified names space that is internally implemented with separate mechanisms. For example, a user interface might either route all names through a single service which forwards requests to private name spaces, or have to obtain information from the name's context or content that indicate where to send it for translation

4.5.3 Recommendations

The following naming features are proposed for the strawman DOS.

- There should be at least two levels of object names:
  - system-level names: immutable, generated on each host, one per each (replicated) object.
  - user-level names: capable of being placed in an organization for convenient reference by users; supported by an efficient name server that maps user-level names to system-level names.

- The types named to maintain type name equivalence between clusters.
- Hosts should be named based on transport layer naming.
- Clusters should be named in a flat space.

The following issues require further examination.

- How are names of different types of entities (objects,hosts, types) handled to allow access from remote clusters?
- What is the relationship between names of entities in the transport layers and those in the session and higher layers?
- Is name translation separated from location binding?
- Is a logically centralized or logically distributed name server used to translate user-level names to system-level names?
- What components of the system handle user-level names: just the user interface, or the user interface and services?

4.6 ADDRESSING

Addressing is the function of referencing an entity by its name for the purpose of accessing it. In object oriented systems, an object is addressed by naming it when invoking an operation. If the system is to support network transparent, host independent, and survivable access to objects, then addressing should not require specifying the host where the object is located, the process that will execute the operation, or whether or not the object is replicated. Binding is a function conventionally provided by the communication layer above the layer that provides message delivery.

The chief purpose of addressing is to bind the name of an object to the location of its state and the implementation of the operations defined on it. A location binding is needed to move an operation invocation on an object to the object itself. An implementation binding is needed to select the specific implementation (i.e., code) of an operation that is named in the invocation.

Bindings of location and implementation can be done statically or dynamically. Static bindings indicate that the binding is done prior to the invocation; when the type operations are compiled, when the type is configured, when the object is created, or when the type is modified after objects are created.

Dynamic binding occurs at invocation time. Location binding must be dynamic if all of the names to be re-bound are not available at binding time. For example, it is not generally possible to rebind all instances of an object's names at the time an object is migrated to a new host because the names may be imbedded in other objects dispersed throughout the system.

Dynamic implementation binding is not usually critical; it trades performance for greater flexibility in determining type implementations. To support adaptability in BM/C3 applications, however, it may sometimes be necessary. Dynamic implementation binding receives considerable attention in object-oriented programming research.

4.6.1 Name to Location Binding

In order to discuss object addressing it is necessary to define several new terms. Call a type implemented in a particular cluster a service. Each service has a domain. The domain of a service is the set of hosts where that service may execute and where objects managed by the service may reside. There are several factors relating to the implementation of the service that have an impact on object addressing. They include:

- how object migration changes name to location binding of objects;
- how service migration changes name to location binding of services;
- whether a service can be implemented across many hosts or on a single host;
- how replicated objects are named;
- the size of the domain of a service: a single host, a cluster, or multiple clusters;

The overriding consideration for location binding is performance: how quickly can the mapping of names to locations be performed in order to transport the invocation request to the server. Systems conventionally provide several levels of optimization techniques built over a name-to-location mapping facility. Optimizations include caching previous bindings in interfaces, client processes, hosts, and name servers; or applying heuristics based on the context in which objects are used (e.g., other objects being accessed) or known properties of the object itself (e.g., where is was created).

When caches and heuristics fail, the binding must be performed by the underlying binding service. Binding information may be provided by a separate naming service designed specifically for that purpose, or within the implementations of the types themselves. Placement of binding information is determined by its frequency of use and the scale of the information. Local area network experiments in Locus and Cronus have demonstrated that reasonable size caches generate hit rates of well over 90%. These results argue that multicasting to the service for a type to find an object when caches fail is feasible in modest sized networks. Larger networks (greater than 100 nodes) or the multicluster environment may require alternative methods.

One alternative is to define a search path for an object. For example, look for the object on the host it was created; if not found, broadcast in its cluster; if not found, send to a lost and found object service. This approach is effective only if most applications of the system have similar properties with respect to object location and migration. This approach might be appropriate for BM/C3 applications.

A second alternative relies on a name server to provide the binding (e.g., Xerox Grapevine). Name servers generally require other services to notify them when they create or migrate an object. This approach creates a possible bottleneck at the name server and requires algorithms to partition the name space efficiently for scalability.

Binding of names to locations is conventionally a multistep process. Using the definition of domains described above, a four step name to location binding process is conceivable: first names map to domains, within domains to a service, within a service to a host, on the host to a particular server process. Other binding

techniques might include further steps (e.g., binding services to implementations) or fewer steps. Each step can make use of caching and heuristics. At each step, either name servers or multicasting can be used. The proper choice depends on the following properties of each level:

- the size of the name space being mapped at each level;
- the physical size of the area being searched; and
- the performance of communication protocols between hosts in different clusters.

One strategy which considerably lessens the cost of location binding across clusters is to embed the cluster identifier in object unique identifiers. This also restricts future migration of the object, however.

Two schemes are commonly used for addressing. One is based on connectionless operation invocation on objects in which each invocation on an object is treated independently of others (with the exception of performance enhancements such as caching object locations or access control bindings). The second model explicitly defines connections which represent a logical session between a client and a service. Addressing occurs at connection creation time only.

Connection-based systems are well-suited for situations in which the connection lives long enough to amortize its creation cost over its lifetime. Connections are particularly useful for bulk data transfer and primarily stream-oriented computations. Applications needing to pipeline intermediate computation results for increased parallelism are examples. A strawman DOS might support connection-oriented facilities in addition to independent invocations.

4.6.2 Other Types of Addressing

Addressing individual objects is not sufficient for set-at- a-time operations involving content-based addressing over multiple objects. Examples of these operations where the object-at-at-time paradigm breaks down are a relational query and operation involving two or more objects (e.g., printing a file on a specific printer). The problem in these examples is that a specific object needs to be designated when the operation is actually on a set of objects. Furthermore, a query language of a database is not object oriented. Mapping of that algebra onto

DOS addressing demonstrates the limitations of object-oriented addressing and shows that the entire database must be treated as a single object.

It would also be useful to be able to invoke an operation on a set of objects of a single type. For example, backing up, restoring and mounting groups of objects (analogous to a file partition) or simple associative access are not functions easily supported by the object model. The traditional approach to this problem is to use class operations, implemented in the Cronus DOS as invocations on a generic object. Each manager of a set of objects of a type implements a generic object. The generic object contains meta-information about the type (e.g., number of objects) and about the status of the particular manager. Generic objects have publicly known names (currently, the type concatenated with a host address). They are used to refer to the type, the process implementing the type, meta-information, or the set of all objects of the type.

Generic objects are a step in the right direction, but fall short because they are not a generalization of a set of objects. A facility may be needed to allow users and applications to group objects into collections. A collection is simply an enumeration of objects of a type, similar to an entity group in VMTP or a group in Grapevine. Instead of invoking an operation on an object, operations can be invoked (interchangeably) on objects or collections. Collections are distinguished from simple object enumerations by their support for associative, content-based creation and use. Operations on collections introduce many of the same problems as distributed query execution in distributed database. As with distributed DBMSs, special session and transport layer protocols may be needed to support their access (e.g., reliable multicast).

4.6.3 Recommendations

The following addressing features are proposed for the strawman DOS.

- Dynamic name to location binding, where object location, replication, and implementation are not specified in the invocation.
- A multilevel binding facility that supports access within and between clusters.

- Use of caching and heuristics at each binding level to improve performance.
- Selection of multicasting or name server approaches at each level based on scale and properties of the DOS.
- Investigate static vs. dynamic name-to-implementation binding.
- Addressing techniques for referencing data which do not conform to a simple object-oriented addressing technique (databases and collections).
- Implementation of service domains to allow applications to execute across cluster boundaries.

4.7 ACCESS CONTROL

Access control is divided into mandatory and discretionary controls. Mandatory controls are primarily concerned with constraining the flow of information in a system, and the rules governing flow are set by a central security authority. Discretionary controls concern the access to information requested by an authenticated accessor. While both mandatory and discretionary controls are relevant to BM/C3 applications, this section is primarily concerned with discretionary access controls.

Access control in an object-oriented system is the check that the accessor of an object has the privilege to access the object with the operation named in the invocation. A mapping is defined between privileges and operations by the developer of the object type. Access control is traditionally discretionary in nature, meaning that the privileges to access an object may be modified in unrestricted ways by a designated group of users. The access matrix model is used to formalize discretionary access control. Rows of the matrix indicate accessors, columns indicate objects that can be accessed, and cells indicate the privilege of the accessor to access the object.

Access controls can be implemented either with capabilities or access control lists (ACLs). Capabilities are formed by grouping matrix cells by row and storing them with the accessor. ACLs group by column and are stored with the

object. The symmetry of these mechanisms makes one approach's advantages the other's disadvantages, and vice versa. This, in turn, tends to polarize people's views on the subject. Some systems combine the two approaches.

The capability approach allows accessors to review all of their privileges, allows privilege transfer between processes, and allows accessors to be given access privileges with finer granularity (e.g., spawned processes can inherit a subset of capabilities; inheriting a subset of identities usually does not make sense). Capability-based systems also commonly use capabilities to name objects at the system level. On the other hand, ACLs make it possible to review all of the potential accessors to objects, makes revocation of privileges trivial, and is the widely adopted convention for access control.

Although capability-based systems have received considerable attention in systems research, there has been little experience with their application in distributed systems that have received extensive use. Supporting capabilities is generally more complex than ACLs because access rights are distributed with the clients rather than grouped where objects are managed. Capabilities must be stored with the client and transferred to services in a way that they cannot be forged or corrupted. It is also difficult to integrate separately developed capability-based systems, or a capability-based system with an ACL-based systems. This reflects the difference in the representation and interpretation of access rights in capabilities, and the common use of capabilities for naming. It has been observed that the ability to review who has access to information in computer systems used by the military is far more critical than the ability to review those privileges a particular user has. All of these factors seem to limit the efficacy of capability-based systems for BM/C3 applications; we will to focus on ACLs for the remainder of this discussion.

The issues that are relevant to access control in a DOS are:

- where are access control checks performed;
- what are the authentication identities used for access control;
- how are the authentication identities transferred to the object's service;
- how are access rights represented, and how expressive are they;
- is access control type specific or type independent;

•how can nested operation invocations be supported;

•how can mandatory access constraint policies be supported;

•how are untrusted workstations handled.

Access controls that allow type-dependent specification of access privileges are far more expressive than systems based on read/write/execute privileges. Effectively, type-dependent access privileges adopt the same level of abstraction for access authorization as it does for object access. Type-dependent access controls can be implemented in one of two ways: in a decentralized fashion within each service, where the integrity of the authorization is only as good as the integrity of the service; or in a logically centralized reference monitor. The reference monitor approach makes the addition of new types and specialization of access control in an application difficult. Furthermore, since objects themselves are replicated, their ACLs need to be replicated as well. Kernel minimality favors placement of ACLs in services.

4.7.1 Access Authorization

An access control list (ACL) is a list of entries, where each entry containing an identity and a set of rights. When a client invokes an operation on an object, the access control facility checks that an authenticated identity of the client (the client identity may include multiple components) appears in an entry in the ACL of the object being accessed, and the rights in the entry map to the operation invoked. The access rights-to- operation mapping function can be a simple one-to-one mapping or significantly more complex.

In an object-oriented system, the rights defined for a type are dependent upon the semantics of the type. Since each type may define a different set of operations, rights and their mapping to operations is type specific. While access control can be table driven, the extensibility of object oriented systems makes access control extremely dynamic and difficult to place in a centralized mediation facility. The solution commonly adopted is to place access control to objects of a service in the service itself.

Placement of ACLs in services makes it difficult to ensure the integrity of access authorization in untrusted user-developed services. These problems relate

to the trustworthiness of services themselves, which is a fundamental issue of user- extensible systems. If an untrusted service is allowed to be configured in a cluster, then the access control to the service is no more reliable than the service itself. Thus, the problem lies not with untrusted access control, but rather with untrusted services. As a result, access control is only as reliable as the service it is implemented in and the host on which the service executes. Many believe that access control only needs to be as reliable as the application, too.

Services implemented in untrusted hosts (e.g., personal workstations) are not tamperproof, and as a result their access control cannot be trusted. The role of untrusted hosts needs to be taken into consideration in the design of access control facilities and configuration of services in a system.

Access control should be supported in a uniform fashion, despite their implementation in a type-dependent way. For example, operations map to rights and access control operations interfaces should be consistent across applications. The distribution of access control among services makes uniformity more difficult to ensure. As with other properties of object- oriented systems, uniformity of aspects of functions across different types is achieved through type inheritance; in the case of access control, inheritance of ACL-related operation specifications.

A potentially important need of BM/C3 applications that has received little attention is nondiscretionary ACLs. Nondiscretionary ACLs are ACL-based constraints that are less flexible than discretionary controls; the policy governing the assignment and modification of access rights is set by a security administrator rather than the creator or owner of each object. Nondiscretionary ACLs would allow the definition of cluster-wide or application-based policies constraining access to objects that cannot be circumvented. Nondiscretionary ACLs could be useful to severely restrict the set of users that may ever invoke critical operations. Further research into the implementation of nondiscretionary controls is needed, but a plausible approach is to separate ACLs into discretionary and nondiscretionary parts and establish different rules for how they may be modified.

4.7.2 Authentication Identities

An authentication identity generally consists of a principal (a user or service) and zero or more aggregate(s) of principals. Examples of aggregates include projects (Multics), groups (Unix), disjoint groups, rights identifiers (VMS), the system (VMS), world, and an arbitrary set of groups (Cronus). Using an arbitrary set of groups that can be enabled and disabled under client control appears to be the most general of these mechanisms.

Although individual object types vary, the patterns of object use across different types of objects are similar with respect to privilege sharing. For example, the ability to monitor components of the system is a role a user may have that is type independent. While a group would define a set of principals, a role would define a set of access rights that a group of principals may have with respect to a set of objects (which specific objects would be determined by the ACL of the objects). Roles are particularly relevant to BM/C3 applications where users can be specifically characterized by the limited ways in which they use the system. Roles are also useful ways of implementing nondiscretionary controls.

The major limitation of ACLs is the difficulty of adequately supporting privilege transfer in nested operation invocations. Consider the case of a client invoking an operation on an object; the service receiving this request, in turn, invokes operations on other objects in order to perform the function. For example, a printer server needs to invoke read operations on a file to be printed. The authority of a service to perform an arbitrary nested invocation is either permanently delegated to the service (e.g., the printer server has read access to all files) or given to the service by the original invoker (the client requesting the read passes a privilege to the printer server).

Permanent privilege vested in the nested caller has two problems. First, the server can accidently circumvent important access controls (the printer server prints a file the client did not have access to). Second, since an arbitrary service may have been implemented by an untrusted user, investing extensive privilege in the server is inappropriate.

The alternative is for the client to pass privileges to the server that allow the server to act safely on his behalf. This type of privilege transfer is precisely what capabilities are effective for; the problem with ACL-based controls is that the client does not hold privileges. Rather, the client is associated with an authenticated identity which is used to perform access control.

Existing ACL-based schemes might need to be extended to support proxies, which are temporary authentication identities that provide their possessor with a limited set of access privileges. Issues to be addressed include how to specify a proxy, how to transfer proxies, and how to limit proxy lifetimes.

Services need to be concerned with the accuracy of authentication identities obtained during access authorization. In a single cluster, authentication identities can be obtained from the trusted service that maintains these identities, and then cached. In the Internet, it is not always possible to determine the trust of any service in a remote cluster. One solution is to record the trustworthiness of each cluster in the system. When a service receives a request from a remote cluster, the authentication identities are treated in a manner commensurate with the trustworthiness of the client's cluster.

4.7.3 Mandatory Security

Mandatory security constrains the flow of information as defined by a security policy. The most widely recognized security policy is a Bell-LaPadula policy, which defines security levels and need-to-know compartments. While both discretionary access controls and nondiscretionary ACLs are concerned with access to objects at their level of abstraction, mandatory security is concerned solely with the flow of information among entities (processes, users, devices) in the system.

A DOS cluster providing mandatory security (MS) may well have a different internal structure than the DOS in other clusters, but that structure will not necessarily be visible outside the cluster. In order to ensure compatibility between an MS cluster and other clusters it is necessary that the MS cluster supports the same interfaces as the other clusters and compatible interconnection

mechanisms. Both DOS interfaces and the transport and session level protocols need to be examined from the standpoint of mandatory security. Issues to be addressed include covert channel analysis and data integrity.

4.7.4 Recommendations

The following access control features are proposed for the strawman DOS.

- Authentication identities should consist of a principal, one or more aggregates of users, and a set of roles.
- Access control should be implemented in a type specific manner inside each service.
- Access control facilities should be expanded to support proxy access and nondiscretionary controls.
- Protocols need to be explored that permit connections of clusters that support mandatory security policies to clusters that do not.
- A service performing access authorization for access requests crossing cluster boundaries must take into consideration the integrity of the accessor authenticated identity based on the trustworthiness of the client's cluster services.

4.8 AUTHENTICATION

Authentication is the assignment of an identity by a trusted agent to an active entity in the DOS. An active entity is most often a process which represents a user or a service, communicates with other processes, and accesses objects. The main purpose of authentication is to give clients an identity which can be used as the basis for deciding whether the client is permitted to access an object. Authentication occurs at least once in the lifetime of a process within a given context, generally when it is first created or first attempts to access an object.

The authentication service must have two properties: continuous operation and high integrity. Continuous operation requires the distribution and replication

of all authentication information. Concurrency control algorithms which do not block access during failure conditions might be advantageous. High integrity is provided by placing authentication services on machines which are secure from tampering and by validating the correctness of the algorithms used by the service.

The design of the authentication service is greatly affected by the internetworking of clusters operating under different administrative authorities. Mechanisms should be developed which allow flexible communication but preserve autonomous control.

There are four issues that relate to authentication in the DOS:

- scope of the authentication,
- the procedure for authenticating different entities within an authentication domain,
- the procedure for authenticating entities in different authentication domains, and
- the placement of the identity established through authentication.

4.8.1 Authentication Scope

The scope of authentication must be at least cluster-wide to allow host transparent access of resources within the cluster. Global authentication expands the scope of authentication beyond the boundaries of clusters, and is possible only if the identity of the client, as established in the client's cluster, is trusted. Since a client in a cluster with a compromised authentication service is able to masquerade as any identity, access control must take into consideration the integrity of the client identity for remote clients. This is considered further in the next section on access control.

The alternative to global authentication is to have each client authenticate itself in each cluster when access occurs. Thus, a client authenticated in the cluster it executes in would need to reauthenticate itself in another cluster if it wished to access an object in that cluster. Such a scheme could be implemented by applying the work of Birrell on Secure Communication Using RPCs [Birrell 1986]. This work relies on encryption to authenticate a client by an authenticator trusted by both the service and the client. Key distribution for client to service

encrypted communication can be distributed as a byproduct of authentication. The problem with this approach is that it requires the client and service to execute in a single authentication domain or the explicit participation of the user (e.g., providing a password) in each authentication domain that a service resides.

4.8.2 Authentication Protocols

In this section only the authentication of users and services in the DOS is considered. Layers underneath the DOS will need to address host authentication and gateway authentication. User authentication occurs when users log into the DOS and identify themselves. Conventional password and one- way encryption schemes are adequate for user authentication.

Service authentication occurs when a service is initiated by the DOS and both its identity and the types of objects it may service are determined. Authentication of a user is generally based on information (e.g., a password) provided by the user that is only possessed by that user. This approach is infeasible with services because of the inability of a service to store information in a safe place without being previously authenticated (a chicken and the egg problem). Services should be legitimately initiated under two conditions:

1. a trusted entity indicates that the service may be initiated; and
2. a trusted entity indicates that the service is permitted to execute on the host.

The first condition is generally satisfied by the system administrator initiating a service or the kernel starting a service in response to an invocation request. The second condition might be satisfied by a Cluster Configuration Service responsible for controlling which services may execute on which hosts.

As described, untrusted workstations would be able to execute processes which masquerade as services. It is unrealistic to expect the software running on a workstation to be tamperproof. The only reasonable approach is to only allow services to execute on the workstation for which tampering is not critical, and never to 'acknowledge' the declaration of a service on a workstation which is not configured to execute on the machine. Since the DOS kernels are responsible for routing invocation requests to services, they must maintain a record of

authenticated server processes which reflect the Configuration Service permissions. No client can expect accurate execution of its requests when it executes on a host whose DOS kernel is compromised.

4.8.3 Identity Placement

The authenticated identity of a client can take many forms; these are the subject of the section on Access Control. The location of these identities must be either in tamperproof kernels or trusted services. An authenticated identity can be associated with an individual process or with a process tree, since all processes initiated by an authenticated process generally inherit (at least a subset of) the creator's identity. Authenticated identities could be maintained by the kernel by the Authentication Service, or by a Process Service which manages process objects.

Kernel maintenance of authenticated identities assumes the trustworthiness of the kernel on each machine. By placing authenticated identifiers in a Process Service, process management can be made independent of machine boundaries. Each trusted host could maintain a Process Server, and the Process Servers for untrusted hosts are placed on trusted hosts. This approach scales well.

Using an Authentication Manager to maintain authenticated identities makes it costly (i.e., requires communication to the remote Authentication Service) to inherit the authenticated identities from a parent process. Relying on either the Authentication Manager or Process Manager scheme introduces fault tolerance problems if the authenticated identities are not replicated, and complexity if they are.

4.8.4 Recommendations

The following authentication features are proposed for the strawman DOS.

- The Authentication Service must operate continuously and have high integrity.

- Authentication should not impair access across cluster boundaries. As a result, authentication should either be global or established as a byproduct of an operation invocation. Access control should limit access based on the integrity of the authentication.

- User authentication should be provided through conventional approaches except where security requirements dictate alternate schemes.

- Service authentication should be performed by the kernel on the host where the service executes in conjunction with a service responsible for the configuration of services in the cluster.

- The placement of authentication information ought to be determined by the system architecture and access control procedures.

4.9 STORAGE MANAGEMENT

In this section, we present in brief, some acceptable storage management techniques for object-based systems, and show the range of features such a storage management technique may be required to handle.

Briefly, storage management is the part of the object based system that handles the storage of the objects themselves. Although, objects are the conceptual entity that are stored and acted upon by such systems, files may exist as the facility that stores objects. There are other criteria of object storage systems that may not apply to file storage systems, such as consistency maintenance, handling of executable code, transparent naming and recovery. Let us now look at these issues in more depth.

4.9.1 Storage Management Techniques

There are two established techniques for storage management of objects used by current distributed operating systems. These two techniques, File System Based Storage and Demand Paging Based Storage are described in the following subsections .

4.9.1.1 File System Based Storage

In the file system method of storage management, each object is stored in one or more files, managed by the file system. A one file implementation contains the entire code and data contained in the object as a sequential file. However an object may contain several logical entities, which can be stored in several files for further efficiency. The advantages of using file systems for object storage are multifaceted. File systems have been around for a long time and have been researched, optimized, and almost perfected. Performance of file systems have been extensively studied and now it is possible to build high performance file systems using several techniques such as buffering, look-ahead, and block placement. There are many undocumented optimizations known to implementors of file systems that extend their performance further.

4.9.1.2 Demand Paging Based Storage

This storage management technique *maps* the objects stored on secondary storage directly into virtual memory. The data and code for the objects are stored on disk, using disk segments, structured similar to files. However, unlike the file system storage technique the access method does not use file access methods, but instead uses demand paging mechanisms to bring the objects into physical memory when needed.

Though demand paging systems have been in use for nearly as long as file systems, file systems are better optimized for sequential access than are demand paging systems. The optimizations used in demand paging are generally limited to pre-paging and swapping out all pages of a program not in execution. Also demand paging has been used and optimized for program execution, rather than for long term storage purposes.

The demand paging storage scheme as described can be used only on hardware supporting demand paged virtual memory. For systems that do not support this facility, the entire object has to be brought into the main memory on invocation. This requires reading the object into memory, and thus the mechanism becomes equivalent to a file based scheme for this machine. In fact, given an efficient implementation of files, the file technique is better suited for simplistic hardware.

The demand paged storage scheme has been used in Archons and Clouds. Both these systems are currently under development and performance of this technique has not been adequately tested.

4.9.2 Other Issues Impacting Storage Management

4.9.2.1 Recovery

It may be desirable for the object storage system to support locking and recovery when a storage fault is detected. (Storage faults may be detected by implementing careful reads and writes.) In a BM/C3 system, this support is clearly necessary for fault tolerance reasons. The locking of an object is often managed by lock managers and synchronization handlers which are not part of the storage system. However, since recovery entails storing the image of an object at certain stages of execution or recovering updated data (updating reliably), this requires cooperation of the storage system. One issue here is the granularity of locking (i.e., page-level vs. record-level vs. object-level).

For recovery purposes, the object could be stored in files with temporal semantics. That is one file may contain the data before a certain operation was performed (the pre-image) and another file contains the data that was updated (the post-image). Upon completion of the operation (or a group of operations) the appropriate image is discarded (depending upon commit or abort). There are several algorithms that handle recovery and any standard scheme can be chosen. The storage manager simply provides the ability to store before and after images and possibly the ability to store logs, and all these can be achieved using files.

The demand paging mechanisms have to be tailored for handling recovery, when objects are supported via paging. This is accomplished by associating more shadow segments with each pageable segment of the objects. The objects are paged in from the object segments, but are paged out (if necessary) to shadow segments. The commit protocols are used to appropriately copy data (or swap pointers) to or from these shadow segments.

Currently, the choice of recovery mechanisms vary from system to system. Some systems implement a version storage mechanism, some use operation and/or

value logging, and some provide for a variety of mechanisms (and depends on the demands of the application) However, none of the aforementioned mechanisms have addressed real-time and security requirements, and this is a real problem with regard to BM/C3 systems.

4.9.2.2 Consistency

The storage management system also has to guarantee (for some applications) the consistency of the data stored in the objects. This requires that the objects themselves be recoverable, and that all updates to the objects go through a paradigm supporting the concept of "commit".

A storage management scheme which supports consistency involves the use of "recoverable segments". As discussed earlier, objects are stored in "segments" which are pieces of contiguous data. In the case of file system storage, the segment is a file, in the case of demand paged storage, the segment is a basic entity, similar to a swap space. Recoverable segments can be implemented in both the schemes by using two segments to implement each logical segment, one containing the before image and the other containing the after-image. A standard commit protocol (such as 2-phase commit) can be used to coordinate the appropriate segments (this will be discussed shortly), and commit logs may be desirable so that each system knows whether an action was committed or not. Note that this scheme supports both consistency and the recoverability of segments (and hence objects).

The most difficult problem for a replication system is ensuring that all the replicas are "consistent" when they are accessed. It may be necessary to support strong, weak (convergence over time), interactive, or short-lived (time dependent) consistency depending on the application. The "available copies" strategy and the "quorum concensus" strategy support strong consistency. These strategies coordinate the access and updates to copies of repicated data (or objects). These strategies ensure that all the accessible copies contain the same information and are functionally identical to each other, and unaccessible copies are made current when they become accessible. Weak consistency is supported on a few systems, but interactive consistency and short-lived consistency are only research topics and have not yet been implemented.

4.9.2.3 Real-time Considerations and Memory Management Functions

It is ideal in a real-time system to guarantee as much physical memory as needed. This would eliminate the need for virtual memory and its associated delays. Unfortunately, in a BM/C3 system there are physical size and weight limitations (particularly on spaced-based components)which make virtual memory techniques a must. Two memory management functions associated with virtual memory which substantially impact performance are: garbage collection, and determining which objects should be swapped out of memory when there is insufficient space to activate newly invoked objects.

Typically, execution of a process is halted while the "garbage collector" traverses the system to free up memory segments This is clearly unacceptable in a BM/C3 system where there are strict timing constraints. One approach would be to collect garbage whenever there are free clock cycles. However, memory may need to be cleaned up before the next free clock cycles come around. There has recently been research into techniques where garbage collection is done in parallel with normal system execution. This seems to be a promising approach, but has not yet addressed large-scale applications.

Another problem occurs when there is insufficient memory to complete a process. Garbage collection may help, but in some situations everything in memory may be an accessible object (i.e., not garbage). At this point, some object in memory must be swapped out to free up some space. The decision algorithm to determine which object to swap out is critical since falling below a certain hit ratio may incur unacceptable delays. In addition, the unpredictability associated with replacement algorithms makes it difficult to guarantee deadlines from a scheduling standpoint.

4.9.3 Recommendations

- Active objects should have a file system based storage manager.
- Passive object implementations should use demand paged storage systems.

- Other methods of object storage (besides paging) need to be investigated for finer granularities of access (i.e., the granularity of access may not always be a page).

- In the case of embedded systems with limited hardware capabilities, the file based storage system and active objects are probably the best choice. This will allow systems to run object based software without the need for demand paged hardware support.

- Some hardware will have to run commonly used operating systems or message-based kernels for compatibility with existing software. The active object and file based storage systems will be the method of choice in this case since it can be implemented "on top of" common software bases, and will allow a higher degree of compatibility.

- For large command sites, the choice of a storage scheme will depend on whether the kernels for the sites are custom. If so, then the kernels should be built to support objects at the outset (rather than supporting objects on top of a set of mechanisms which support other functions). In this case the passive object and demand paged storage method will be more suitable.

- Other types of support for fault tolerance besides recoverable segments should be investigated (e.g., stable storage, locking segments in memory, etc.).

- Fault tolerance mechanisms which address real-time and security requirements need to be developed.

- A garbage collection scheme which takes into account strict timing constraints must be investigated.

SECTION 5. - CONCLUSIONS AND RECOMMENDATIONS

5.1 CONCLUSIONS

After comparing BM/C3 requirements and the status of current DOS research, we conclude that no current DOS designs adequately address BM/C3 requirements and additional research should be focused on meeting BM/C3 needs. In particular, significant research should be directed toward evolving a DOS design that can simultaneously provide

- Evolvability
- Fault Tolerance
- Multi-Level Security, and
- Real-Time Processing Support

Properly supporting BM/C3 applications requires new methods of system-wide resource management. Generally, existing DOS designs have not significantly evolved beyond their time-sharing network origins. As a result, they still manage system resources to achieve maximum system throughput, with some degree of fairness so that no task has to wait forever. Furthermore, each system element is managed relatively independently. This is not adequate for BM/C3 applications. A BM/C3 DOS must coordinate the use of all system resources to meet real-time response, fault tolerance and multi-level security objectives. This requires new strategies for managing each of the system components.

In particular, we found that current communication system designs could not adequately support the real-time performance requirements of BM/C3. What is needed is a method for the communication system and the DOS to interact to establish goals for delivering information. The communication system must then manage its resources to meet all of the system objectives. The static priority systems used in current communication system designs are not adequate for this. Scheduling procedures that can deal with time-dependent criticality (as discussed in Appendix B) are needed.

5.2 RECOMMENDATIONS

As a first step, RADC should support a thorough analysis of BM/C3 requirements for DOS interoperability support. This work should produce concrete examples of BM/C3 applications to act as benchmarks to measure the adequacy of DOS designs.

The appropriate interface between the application and the DOS needs to be determined. For example, BM/C3 system architects need ways of communicating their desires for fault tolerance. They need to be able to specify to the DOS the tolerance strategy required for each type of fault and each major system function. Similarly,they need ways of expressing real-time performance and security requirements in enough detail to insure that the DOS provides appropriate support.

RADC should sponsor research into DOS designs that simultaneously support multi-level security, real-time performance and fault tolerance. In many ways, these are all conflicting objectives. For example, maintenance and coordination of redundant objects and threads generally improves fault tolerance, but decreases real-time performance. However, in some cases, replicating tasks can help in meeting guarantees for real-time response. The interactions among multi-level security, real-time performance and fault tolerance are poorly understood and deserve further study.

We have identified many specific guidelines and research recommendations, as discussed in earlier sections of this report and summarized here. The guidelines should be presented to current RADC contractors, who should either adopt them or suggest modifications. The research topics recommended here are critical for the advancement of BM/C3 DOS technology, and therefore should be actively pursued.

The voluntary adoption of standards has proven successful in promoting system interconnections in the past. Unfortunately, it is still too early to choose specific interoperability standards since the full consequences of many of the protocols and mechanisms is not well understood and many existing solutions are not geared toward BM/C3 environments. Time did not permit an exhaustive study

of all the DOS components; however, the DOSIRP panel felt the development of a basic system architecture and a set of core components (which are critical to achieving interoperability, and) which could be tailored to address BM/C3 requirements would be a very useful contribution. Increased research to tailor these functions to BM/C3 systems will be a major step towards interoperability.

The nine core components which were addressed include (alphabetically):

Access Control	Naming
Addressing	Storage
Authentication	Synchronization
Communications	System-wide Resource Management
	Timing

For each of these core components it is too early to specify standards. However, it is not too early in some of the areas to suggest tentative design guidelines. In those areas where even guidelines would be premature, some recommended research approaches to address the major issues are given. What follows is a detailed list of guidelines and research recommendations corresponding to each of the core components. What is needed is to determine which of the protocols, mechanisms and ideas discussed in the preceding chapters will provide the necessary performance in a complete, integrated BM/C3 system.

5.2.1 General

Guidelines

- The semantics of a distributed, object-based system should be adopted because it appears to have the greatest potential for dealing with the large-scale, heterogeneous nature of BM/C3 systems. We have suggested in Sections 3.0 and 4.0, a general style which could be adopted and refined.

Research Recommendations

- Analyze in detail the BM/C3 interoperability needs/requirements and tradeoffs with respect to real-time performance, security, fault tolerance and DOSs. In particular, it is important to obtain an objective intermediary who can compile the needs/requirements as viewed by current DOS implementors.

- Encourage experiments in integration. These experiments should look at solutions in both a system context, and more limited experiments such as integrating fault tolerance solutions and real-time scheduling. The information gained in such experiments is essential to evolving an integrated BM/C3 DOS.

- Resolve conflicts regarding the precise semantics of objects. This should include a preliminary study to determine whether the existing differences in semantics are significant or incidental. A scientific approach where each of the semantics are used in controlled experiments (using benchmarks) to evaluate and compare them is one possibility.

- Investigate the important open issues of object-based systems with respect to BM/C3 systems (i.e., real-time, fault tolerance and security constraints). This includes investigation of the implementation issues, inheritance, object naming, parameter types, binding, common operators, user interfaces.

- Determine the benefits of using low-level mechanisms vs. specification of a facility when implementing the system. Where possible, candidate mechanisms should be identified. Then experimental approaches leading to a *better understanding of these mechanisms are needed.*

- Encourage research on, and creation of services, facilities, and protocols based on policy/mechanism approaches, instead of the ubiquitous monolithic approaches used now.

5.2.2 System-wide Resource Management

Research Recommendations

- Develop common object management functions. e.g., create, delete, replicate, assign_values, etc. This includes determining what the DOS needs are, then making tradeoff studies, and then seeing if a common set emerges.

- Devote significant effort to system-wide resource allocation and scheduling approaches and algorithms where timing constraints, task criticalness and overloads are important. Determine the implications on the system structure.

5.2.3 Communications

Guidelines

- A reliable RPC mechanism is needed that has the following features: multicast, real-time messages, secure messages, forwarding, and streams.

Research Recommendations

- Determine the conditions under which it would be advantageous to incorporate asynchronous communications primitives.
- Study the applicability of protocols such as VMTP, and protocol generators such as Chesson's work in a BM/C3 environment concentrating on real-time performance, fault tolerance, security and adaptability.
- Develop a fault tolerant lightweight communications protocol that does not have the shortcomings of existing protocols and takes into account real-time constraints.
- Determine which methodology is appropriate for development of the communications service: starting with low-level candidate transport mechanisms, or developing a communications facility for BM/C3. Answer questions such as: what is the proper set of message types which need to be supported, how can those types evolve, what are the communication interfaces and primitives, etc.

5.2.4 Timing

Research Recommendations

- Develop standard time service interfaces for Delay, GetTime, SetTime and Wakeup functions.
- Develop efficient time synchronization techniques for BM/C3 systems.
- Study the impact of having loose or even no time synchronization in BM/C3 systems.

5.2.5 Synchronization

Guidelines

- Timeouts and agreement should be used for failure detection.

Research Recommendations

- Study is needed regarding synchronization techniques in the context of distributed, fault-tolerant, real-time BM/C3 systems. The two-phase commit protocol, atomic broadcast and Byzantine Agreement protocols must be rethought for BM/C3 environments.

- Develop a form of transactions appropriate for BM/C3 applications. Various models need to be tested under different BM/C3 situations to determine which are best.

5.2.6 Naming:

Guidelines

- At least two levels of objects names are needed: user level names and system level names.
    - System-level names should be globally unique names which are immutable.
    - User level names should be hierarchically organized and supported by an efficient name server.

- Types should be named to maintain type name equivalence between clusters.

- Host names should be based on the transport layer naming.

Research Recommendations

- Investigate the tradeoffs of a flat vs. a hierarchical name space for clusters. (Hierarchical name spaces have been useful in real-time systems.)

5.2.7 Addressing

Guidelines

- Use caching and heuristics at each binding level to improve performance.
- Base the selection of multicasting (or other name server approaches, e.g., unicast, broadcast) at each level on the scale and the properties of the DOS.
- Implementation service domains to allow applications to execute across cluster boundaries.

Research Recommendations

- Clarify the advantages and disadvantages of static vs. dynamic name-to-location bindings in a BM/C3 context.
- Develop a multilevel binding facility that supports access within and between clusters.

5.2.8 Access Control

Guidelines

- Authentication identities should consist of a principal, one or more aggregates of users, and a set of roles.
- Access control should be implemented in a type specific manner inside each service.
- The access authorization service for intercluster access requests must take into account the integrity of the client's (local) cluster authentication service.

Research Recommendations

- Study in more detail the tradeoffs between capabilities and ACLs with respect to BM/C3 systems. ACLs with caches seem more suitable than

capabilities for BM/C3 systems because they more closely match military doctrine.

- Expand access control facilities to support proxy access and nondiscretionary controls.
- Explore protocols that permit connections of clusters that support mandatory security policies to cluster that do not.

5.2.9 Authentication

Guidelines

- Use of global authentication or authentication established as a byproduct of an operation invocation is recommended.
- The authentication service must operate continuously and have high integrity.
- A user and a service authentication protocol is needed.
    - A password and one-way encryption is acceptable for user authentication unless security requirements dictate alternate schemes.
    - Service authentication should be performed by the kernel on the host where the service executes in conjunction with a service responsible for the configuration of services in the cluster.

Research Recommendations

- Determine the where the placement of authentication information ought to be, based on the system architecture and access control procedures.

5.2.10 Storage

Guidelines

- Passive objects should be implemented by demand paging if a customized kernel is designed to directly support objects.

Research Recommendations

- Investigate other methods for object storage, particularly if the granularity of access is not a page.
- Investigate support for fault tolerance and time constraints. e.g., recoverable segments, locking segments in memory, timing considerations in fast virtual memory or not using virtual memory at all, etc.
- Develop garbage collection techniques that are appropriate for a BM/C3 environment.

APPENDIX A - WORKSHOP DETAILS

The project began with an electronic mail forum and initial survey, followed by a two-day workshop in July and another two-day workshop in August. Both workshops were held at the DRC offices in Rome, NY. Following each workshop, the participants prepared written discussions of crucial issues and exchanged comments by electronic mail. The project participants are listed below.

PARTICIPANTS	INSTITUTION	PROJECT
Professor John A. Stankovic (Moderator)	University of Massachusetts - Amherst	Spring Project
Professor David R. Cheriton	Stanford University	V System
Professor Partha Dasgupta	Georgia Institute of Technology	Clouds
Mr. Mike Dean	Bolt, Berenek and Newman	Cronus
Professor E. Douglas Jensen	Carnegie-Mellon University	Alpha/Archons
Dr. Thomas Raeuchle	Honeywell, Inc.	HOPS
Dr. Richard C. Schantz	Bolt, Berenek and Newman	Cronus

Figure A-1 List of Participants

Also, our thanks to Dr. Jonathan Silverman, Honeywell, and Dr. Steve Vinter, Bolt, Berenek, and Newman, for contributing to the research project.

APPENDIX B - BM/C3 SYSTEM CHARACTERISTICS AND REQUIREMENTS FOR DOS SUPPORT

B.1 PURPOSE

The purpose of this discussion is to describe the general characteristics of a BM/C3 system and to relate these characteristics to DOS support requirements. Current BM/C3 systems rely primarily on humans for information processing with relatively little computer assistance. To meet more demanding requirements and provide faster response, computers must carry much more of the information processing load. Yet, many of the characteristics of a human organization, which we often take for granted, must be preserved. This section attempts to explicitly describe some of these characteristics.

B.2 HIERARCHICAL STRUCTURE

A BM/C3 system has a hierarchical organization corresponding to the a military command structure or a business organization. As you move down the hierarchy the scope of control narrows, the objectives become specific and immediate, and the options for response become more limited.

At the top of the structure, the overall commander (Commander-in-Chief or Chief Executive Officer) assesses the overall situation and sets goals for his subordinates. He generally does not deal with detailed information or issue detailed instructions. His time scale for reaction is relatively slow.

At intermediate levels of the structure, middle-managers (division commanders, departments managers) have a particular functional responsibility and/or limited spatial area of responsibility. Their primary activity is monitoring the situation, adjusting goals and priorities for their subordinates, and allocating resources. Their reaction time needs to be at least an order of magnitude faster than the ultimate commander.

At the bottom of the command structure, the "foot soldier" deals with a limited portion of the environment and has a limited objective specified in some detail. He has limited options for reaction and must react on a time scale that is orders of magnitude faster than the ultimate commander.

To automate many of these functions, we need a wide range of services from the DOS. Lower levels in the system will need real-time processing services, but may not need direct security support. Higher-levels may have less need for real-time processing services, but will definitely need excellent security mechanisms.

B.3 FUNCTIONAL ASSIGNMENT AND LOAD BALANCING

In a human organization, functional assignments are relatively static. An accountant cannot immediately become a production manager. A bank teller cannot immediately become a loan officer. An actor cannot immediately become President.

Within functional areas, tasks are fairly readily shared among servers. Any bank teller can handle a simple deposit. Thus, the task load can be easily and quickly balanced within a functional area, but balancing the load across functional areas is more difficult and slow.

Because of physical constraints and communication costs, the assignment of functions to a cluster of computers will probably be static, but the assignment of activities within a cluster can be dynamic. For example, a sensor satellite may contain a cluster of computers to perform signal processing and sensor control functions. These functions could be allocated dynamically within the cluster, but the cluster would never be expected to take on weapon control functions for another satellite.

Their are several obstacles to load balancing across functional areas. First, the location of physical assets (such as radars or computers) cannot be readily changed. Second, the larger the distance between physical objects, the greater the limitations on data transfer. Third, the code to implement complex functions is too large to be rapidly transferred between locations or to be replicated at each location.

Therefore, it makes sense to design a system with a cluster of computers near a physical asset and to statically allocate associated functions to that cluster. Load balancing is desirable within a cluster, but is not feasible among clusters.

B.4 ACCESS CONTROL

The system architects establish the behavior of a BM/C3 system by prescribing the behavior of each class of object in the system and specifying their modes of interaction. They prescribe and control the behavior of people within the system by training and by written policies and doctrine.

A BM/C3 system can have many modes of operation. These include:

- Standby (peacetime)
- Alerts
- Exercises and tests
- Battle

Of these, the most demanding may actually be exercises and tests. This is because not only the heavy demands of wartime must be simulated, but also stimulus generation, environment simulation, performance monitoring and data capture must be supported, and tight security must be maintained.

It is extremely important that the command authority maintain tight control to prevent the inadvertent commencement of hostilities, to limit the engagement and to guarantee prompt cessation of hostilities. One thing this implies is that in peacetime weapon release authority resides very high in the chain of command. During battle, weapon release authority moves the bottom of the command chain, but this authority can be revoked at any time. This implies a changeable access structure for the routing of signals between objects and different behavior patterns for the objects themselves, depending on the mode of operation.

To maintain order and positive control, communication between elements of a BM/C3 system should be strictly regulated. Orders follow the command hierarchy. A private in one army does not take orders from a private in another army, but direct communication between them for a specific purpose may be necessary. An example is an infantry man acting as an artillery spotter. He communicates

information needed to correct the aim, but he does not order the fire.Another example is a direct link between a surveillance radar and a guided weapon or aircraft to provide corrections to insure intercept of a target.

The implication for DOS design is that a few fairly static access structures are needed to support the different modes of system operation. For a particular mode, the access structure may be changed occasionally, but not very often. A shift from one mode of operation to another needs to occur rapidly and under positive control. For security, changing the access structure needs to be a well guarded operation. The system architect should be able to prescribe all allowable interconnections of objects in the system on a non-real-time basis. The DOS should provide a mechanism to manage access in real time. This also contributes to fault tolerance, by providing "firewalls" or "watertight compartments" to limit damage that could be caused by faulty software.

B.5 CONCURRENCY, ASYNCHRONOUS OPERATION AND SCHEDULING

Activities in a BM/C3 system are viewed as occurring concurrently and asynchronously. A commander issues orders, waits for an acknowledgement and then goes on to something else, expecting the orders to be carried out. If the subordinate cannot complete the orders, he is expected to notify his commander, and he may also notify his commander when the orders are successfully completed. The commander will check back after a specified time to make sure the subordinate is still in existence and perhaps determine why the order has not yet been completed.

The implication for DOS design is that a very flexible task coordination and scheduling approach is needed. Some of the requirements are discussed next.

B.5.1 RESOURCE ALLOCATION

The importance of tasks may be highly time- dependent or spatially-dependent. For example, consider a missile attacking a ship. Tasks associated with tracking the missile may be moderately important, then increase to very high importance as the missile nears the ship, and finally drop to zero importance after the missile has missed (or hit) the ship (see Figure B-1). Once a task's importance drops to zero, it should be purged from the system.

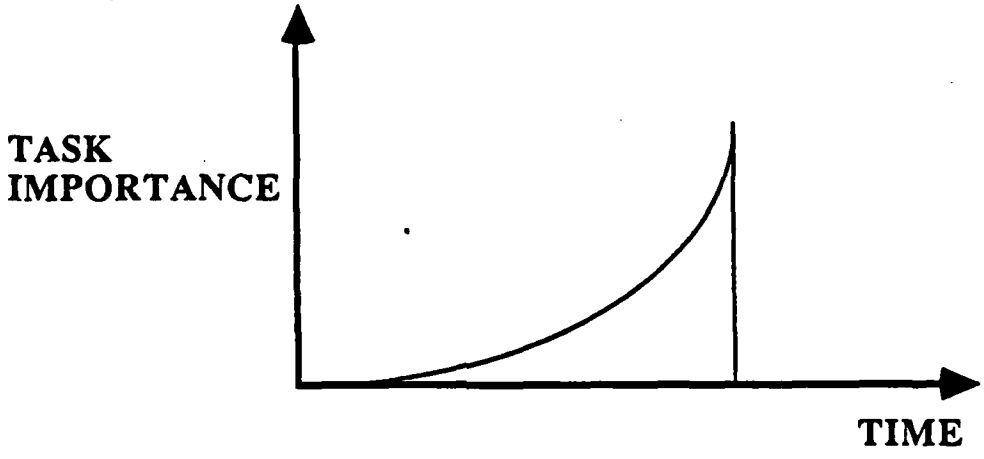


Figure B-1 Task Importance vs. Time

From the point of view of the application, it would be most desirable for the operating system to schedule tasks and manage storage based on a value function that follows a time schedule (like the one above) or based on varying properties of objects (such as position). At a minimum, the application should be able to request a change in priority of all tasks associated with a particular object.

B.5.2 PRECEDENCE, PREEMPTION AND PURGING

This is related to the priority issue. The application should have the ability to reorder the precedence of tasks based on the changing situation. This includes "clearing the decks for action" by suspending running tasks and purging the system. This also has a relationship to fault-tolerance. Fault-tolerance can be provided by cloning tasks to run in parallel or by providing alternate approaches (contingency) plans as backups. For most efficient operation, backup tasks should be preempted and purged from the system once one of the redundant tasks

successfully completes. Preemption is also needed to rid the system of orphan tasks. For example, a task associated with a weapon that was just been blown away should be purged from the system. This could be handled by a configuration-manager object issuing a purge command to the DOS.

B.5.3 FLEXIBLE TASK-DEPENDENCY ARCHITECTURE

In many cases, we want a task to execute as soon as it has all of its necessary data or after all of a certain set of tasks has executed. However, after a certain period of time the task should "go with what it's got." For example, the conditions for starting a task might be expressed in a table as follows:

Predicted Time Until Impact	Condition for Task Initiation
1000 sec	A and B and C completed
500 sec	(A & B) or (B & C) completed
100 sec	A or B or C completed
10 sec	Go unconditionally

Figure B-2 Flexible Task-Dependency

B.5.4 TIME DEPENDENT SCHEDULING

Some tasks may want to leave a "wake-up call" and then "go to sleep" until it is time for them to execute. This is essentially the "tickler file" concept.

B.6 DATA FUSION AND INFORMATION VALUE

Information from a variety of sources must be rapidly and properly combined. To prevent confusion, data should be time-tagged. Also the source of the data and the confidence in the information needs to be known. Furthermore, the relative priority for the processing of the data must be determined based on its content.

DISTRIBUTION LIST

addresses	number of copies
RADC/COTD ATTN: Thomas F. Lawrence Griffiss AFB NY 13441-5700	10
Dynamics Research Corporation 60 Frontage Road Andover MA 01810	5
RADC/DOVL Technical Library Griffiss AFB NY 13441-5700	1
Administrator Defense Technical Info Center DTIC-FDA Cameron Station Building 5 Alexandria VA 22304-6145	2
Strategic Defense Initiative Office Office of the Secretary of Defense Wash DC 20301-7100	2
RADC/COTD Building 3, Room 16 Griffiss AFB NY 13441-5700	1
AFCSA/SAMI ATTN: Miss Griffin 10363 Pentagon Washington DC 20330-5425	1
HQ USAF/SCTT Pentagon Washington DC 20330-5190	1

SAF/AQSC
Pentagon 4D-267
Washington DC 20330-1000 1

Director
DMAHTC/SDSIM
Washington DC 20315-0030 1

Director, Information Systems
OASD (C3I)
Room 3E187
Pentagon
Washington DC 20301-3040 1

Fleet Analysis Center
GIDEP Operations Center
ATTN: Mr. E. Richards
Code 30G1
Corona CA 91720 1

HQ AFSC/XTKT
Andrews AFB DC 20334-5000 1

HQ AFSC/XTS
Andrews AFB MD 20334-5000 1

HQ AFSC/XRK
Andrews AFB MD 20334-5000 1

HQ SAC/SCPT
OFFUTT AFB NE 68113-5001 1

DTESA/RQE
ATTN: Mr. Larry G.McManus
Kirtland AFB NM 87117-5000 1

HQ TAC/DRIY 1
ATTN: Mr. Westerman
Langley AFB VA 23665-5001

HQ TAC/DOA 1
Langley AFB VA 23665-5001

HQ TAC/DRCA 1
Langley AFB VA 23665-5001

ASD/ENEMS 2
Wright-Patterson AFB OH 45433-6503

SM-ALC/MACEA 1
ATTN: Danny McClure
BLDG Z50N
McClellan AFB CA 95652

ASD/AFALC/AXAE 1
ATTN: W. H. Dungey
Wright-Patterson AFB OH 45433-6533

WRDC/AAAI 1
Wright-Patterson AFB OH 45433-6533

AFIT/LDEE 1
Building 640, Area B
Wright-Patterson AFB OH 45433-6583

WRDC/MLPO 1
Wright-Patterson AFB OH 45433-6533

WRDC/MLTE
Wright-Patterson AFB OH 45433 1

WRDC/FIES/SURVIAC
Wright-Patterson AFB OH 45433 1

AAMRL/HE
Wright-Patterson AFB OH 45433-6573 1

Air Force Human Resources Lab
Technical Documents Center
AFHRL/LRS-TDC
Wright-Patterson AFB OH 45433 1

AFHRL/OTS
Williams AFB AZ 85240-6457 1

AUL/LSE
Maxwell AFB AL 36112-5564 1

HQ Air Force SPACECOM/XPYS
ATTN: Dr. William R. Matoush
Peterson AFB CO 80914-5001 1

Defense Communications Engr Center
Technical Library
1860 Wiehle Avenue
Reston VA 22090-5500 1

C3 Division Development Center 2
Marine Corps
Development & Education Command
Code DIOA
Quantico VA 22134-5080

AFLMC/LGY 1
AF Logistics Management Center
Chief, System Engineering Division
Gunter AFS AL 36114

US Army Strategic Defense Command 1
DASD-H-MPL
PO Box 1500
Huntsville AL 35807-3801

Commanding Officer 1
Naval Avionics Center
Library
D/765
Indianapolis IN 46219-2189

Commanding Officer 1
Naval Training Systems Center
Technical Information Center
Building 2068
Orlando FL 32813-7100

Commanding Officer 1
Naval Ocean Systems Center
Technical Library
Code 96423
San Diego CA 92152-5000

Commanding Officer 1
Naval Weapons Center
Technical Library
Code 3433
China Lake CA 93555-6001

Superintendent 1
Naval Post Graduate School
Code 1424
Monterey CA 93943-5000

Commanding Officer 2
Naval Research Laboratory
Code 2627
Washington DC 20375-5000

Space & Naval Warfare Systems COMM 1
PMW 153-3DP
ATTN: R. Savarese
Washington DC 20363-5100

Commanding Officer 2
US Army Missile Command
Redstone Scientific Info Center
AMSMI-RD-CS-R (Documents)
Redstone Arsenal AL 35898-5241

Advisory Group on Electron Devices 2
Technical Info Coordinator
ATTN: Mr. John Hammond
201 Varick Street - Suite 1140
New York NY 10014

Los Alamos Scientific Laboratory 1
Report Librarian
ATTN: Mr. Dan Baca
PO Box 1663, MS-P364
Los Alamos NM 87545

Rand Corporation 1
Technical Library
ATTN: Ms. Doris Helfer
PO Box 2138
Santa Monica CA 90406-2138

AEDC Library 1
Technical Reports File
MS-100
Arnold AFS TN 37389-9998

USAG 1
ASH-PCA-CRT
Ft. Huachuca AZ 85613-6000

1839 EIG/EIET 1
ATTN: Mr. Kenneth W. Irby
Keesler AFB MS 39534-6348

JTFPO-TD 1
Director of Advanced Technology
ATTN: Dr. Raymond F. Freeman
1500 Planning Research Drive
McLean VA 22102

HQ ESC/CWPP 1
San Antonio TX 78243-5000

AFEWC/ESRI 3
San Antonio TX 78243-5000

485 EIG/EIR 1
ATTN: M Craft
Griffiss AFB NY 13441-6348

ESD/XTP 1
Hanscom AFB MA 01731-5000

ESD/ICP 1
Hanscom AFB MA 01731-5000

ESD/AVSE 2
Building 1704
Hanscom AFB MA 01731-5000

HQ ESD SYS-2 1
Hanscom AFB MA 01731-5000

Software Engineering Institute 1
Joint Program Office
ATTN: Major Dan Burton, USAF
Carnegie Mellon University
Pittsburgh PA 15213-3890

Director 1
NSA/CSS
T513/TDL
ATTN: Mr. David Marjarum
Fort George G. Meade MD 20755-6000

Director
NSA/CSS
W166
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS
DEFSMAC
ATTN: Mr. James E. Hillman
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS
R5
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS
R8
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS
S21
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS
R523
Fort George G. Meade MD 20755-6000

2

DOD Computer Security Center
C4/TIC
9800 Savage Road
Fort George G. Meade MD 20755-6000

1

Honeywell - SSDC
ATTN: Mr. Jeremy Norton
10009 Boone Ave
MS: Mn63-C040
Golden Valley MN 55427

1

SDI/S-P1-BM
ATTN: Cmdr Korajo
The Pentagon
Wash DC 20301-7100

1

SDIO/S-Pl-BM
ATTN: Capt Johnson
The Pentagon
Wash DC 20301-7000

1

SDIO/S-Pl-BM
ATTN: Lt Col Rindt
The Pentagon
Wash DC 20301-7100

1

IDA (SDIO Library)
ATTN: Mr. Albert Perrella
1801 N. Beauregard Street
Alexandria VA 22311

1

SAF/AQSD
ATTN: Maj M. K. Jones
The Pentagon
Wash DC 20330

1

AFSC/CV-D
ATTN: Lt Col Flynn
Andrews AFB MD 20334-5000

1

HQ SD/XR
ATTN: Col Heimach
PO Box 92960
Worldway Postal Center
Los Angeles CA 90009-2960 1

HQ SSD/CNC
ATTN: Col O'Brien
PO Box 92960
Worldway Postal Center
Los Angeles CA 90009-2960 1

HQ SD/CNCI
ATTN; Col Collins
PO Box 92960
Worldway Postal Center
Los Angeles CA 90009-2960 1

HQ SD/CNCIS
ATTN; Lt Col Pennell
PO Box 92960
Worldway Postal Center
Los Angeles CA 90009-2960 1

ESD/AT
ATTN: Col Ryan
Hanscom AFB MA 01731-5000 1

ESD/ATS
ATTN: Lt Col Oldenberg
Hanscom AFB MA 01731-5000 1

ESD/ATN
ATTN: Col Leib
Hanscom AFB MA 01731-5000 1

AFSTC/XPX (Lt Col Detucci)
Kirtland AFB NM 87117 1

USA SDC/DASD-H-SB (Larry Tubbs)
P. O. Box 1500
Huntsville AL 35807 1

AFSPACECOM/XPD 1
ATTN: Maj Roger Hunter
Peterson AFB CO 80914

GE SDI-SEI 1
ATTN: Mr. Ron Marking
1787 Century Park West
Bluebell PA 19422

MITRE Corp 1
ATTN: Dr. Donna Cuomo
Bedford MAS 01730

SSD/CNI 1
ATTN: Lt Col Joe Rouge
P. O. Box 92960
Los Angeles AFB CA 90009-2960

NTB JPO 1
ATTN: Maj Don Ravenscroft
Falcon AFB CO 80912

Ford Aerospace Corp 1
c/o Rockwell International
ATTN: Dr. Joan Schulz
1250 Academy Park Loop
Colorado Springs CO 80910

Essex Corp 1
ATTN: Dr. Bob Mackie
Human Factors Research Div
5775 Dawson Ave
Goleta CA 93117

Naval Air Development Ctr 1
ATTN: Dr. Mort Metersky
Code 30D
Warminster PA 18974

RJO Enterprises 1
ATTN: Mr. Dave Israel
1225 Jefferson Davis HY
Suite 300
Arlington VA 22202

GE SDI SEI 1
ATTN: Mr. Bill Bensch
1707 Century Park West
Bluebell PA 19422

HQ AFOTEC/OAHS 1
ATTN: Dr. Samuel Charlton
Kirtland AFB NM 87117

ESD/XTS 1
ATTN: Lt Col Joseph Toole
Hanscom AFB MA 01731

SDIO/ENA 1
ATTN: Col R. Worrell
Pentagon
Wash DC 20301

USA-SDC CSSD-H-SBE 1
ATTN: Mr. Doyle Thomas
Huntsville AL 35807

HQ AFSPACECOM/DOXP 1
ATTN; Capt Mark Terrace
Stop 7
Peterson AFB CO 80914

BBN Systems & Technology 1
ATTN: Dr. Dick Pew
70 Fawcett St
Cambridge MA 02138

ESD/XTI 1
ATTN: Lt Col Paul Monico
Hanscom AFB MA 01730

CSSD-H-SB 1
ATTN: Mr. Larry Tubbs
Commander USA SDC
PO Box 1500
Huntsville AL 35807

USSPACECOM/J5B 1
ATTN: Lt Col Harold Stanley
Peterson AFB CO 80914

NTB JPO 1
ATTN: Mr. Nat Sojouner
Falcon AFB CO 80912

RADC/COT 1
ATTN: Mr. Ronald S. Raposo
Griffiss AFB NY 13441

Bonnie McDaniel, MDE 1
313 Franklin St
Huntsville AL 35801

The Aerospace Corporation 5
ATTN: Mr. George Gilley
M1-046
PO Box 92957
Los Angeles CA 90530

AF Space Command/XPXIS 1
Peterson AFB CO 80914-5001

AFOTEC/XPP 1
ATTN: Capt Wrobel
Kirtland AFB NM 87117

Director NSA (V43) 1
ATTN: George Hoover
9800 Savage Road
Ft George G. Meade MD 20755-6000

SSD/CNIR 1
ATTN: Capt Brandenburg
PO BOX 92960-2960
LOS ANGELES CA 90009-2960

Advanced System Technologies
ATTN: Duane R. Ball
5113 Leesburg Pike, Suite 514
Falls Church VA 22041 1

Odyssey Research Associates, Inc.
ATTN: Doug Weber
301A Harris B. Dates Dr
Ithaca NY 14850-1313 1

SRI International
ATTN: Teresa Lunt, BN169
333 Ravenswood Ave
Menlo Park CA 94025 1

SRI International
ATTN: Mathew Morgenstern, BN 162
333 Ravenswood Ave
Menlo Park CA 94025 1

George Mason University
ATTN: Prof Sushil Jajodia
ISSE Department
4400 University Drive
Fairfax VA 22030-4444 1

GE (Advanced Technology Labs)
ATTN: L. D. Alexander
Bldg 145-2, Route 38
Moorestown NJ 08057 1

Concurrent Computer Corp
ATTN: E. Douglas Jensen
1 Technology Way
Westford MA 01886 1

Xerox Advanced Info Technology
ATTN: Barbara Blaustein
7900 West Park Dri, Suite 400
McLean VA 22102 1

Dove Electronics, Inc.
ATTN: John Dove
227 Liberty Plaza
Rome NY 13440 1

W. W. Chu Associates 1
ATTN: Dr. Wesley Chu
16794 Charmel Lane
Pacific Palisades CA 90272

U. S. Army CECOM 1
ATTN: Lakshmi V. Rebbapragada
Center for C3 Systems
AMSEL-RD-C3-IR
Ft Monmouth NJ 07703

M/A 478 1
NASA-Langley Research Center
ATTN: Nicholas D. Murray
Hampton VA 23665

Trusted Information Systems, Inc. 1
ATTN: Steve Walker
3060 Washington Rd
Glenwood MD 21738

Advanced Decision Systems 1
ATTN: Andrew Cromerty
1500 Plymouth St
Mountain View CA 94043

Gemini Computers Inc. 1
ATTN: Dr. Roger Schell
ICS Division
PO Box 222417
Carmel CA 93922

Naval Ocean Systems Center 1
ATTN: Les Anderson
271 Catalina Blvd, Code 413
San Diego CA 92151

Carnigie Mellon University 1
Dept of Computer Science
ATTN: Ray Clark
Schenley-Wean Hall
Pittsurgh PA 15213-3890

Univ of Maryland 1
Dept of Computer Science
ATTN: Ashok K. Agrowola
College Park MD 20742

The Mitre Corporation 1
ATTN: Myra Jean Prelle
Burlington Rd
Bedford MA 01730